SQL on Azure Succinctly

by Parikshit Savjani

Syncfusion | Technology Resource Portal

SQL on Azure Succinctly

By
Parikshit Savjani

Foreword by Daniel Jebaraj

Important licensing information. Please read.

Technical Reviewer: Stephen Haunts
Copy Editor: Darren West, content producer, Syncfusion, Inc.
Acquisitions Coordinator: Hillary Bowling, online marketing manager, Syncfusion, Inc.
Proofreader: Tres Watkins, content development manager, Syncfusion, Inc.

Table of Contents

The Story behind the *Succinctly* Series of Books

Daniel Jebaraj, Vice President
Syncfusion, Inc.

Staying on the cutting edge

As many of you may know, Syncfusion is a provider of software components for the Microsoft platform. This puts us in the exciting but challenging position of always being on the cutting edge.

Whenever platforms or tools are shipping out of Microsoft, which seems to be about every other week these days, we have to educate ourselves, quickly.

Information is plentiful but harder to digest

In reality, this translates into a lot of book orders, blog searches, and Twitter scans.

While more information is becoming available on the Internet and more and more books are being published, even on topics that are relatively new, one aspect that continues to inhibit us is the inability to find concise technology overview books.

We are usually faced with two options: read several 500+ page books or scour the web for relevant blog posts and other articles. Just as everyone else who has a job to do and customers to serve, we find this quite frustrating.

The *Succinctly* series

This frustration translated into a deep desire to produce a series of concise technical books that would be targeted at developers working on the Microsoft platform.

We firmly believe, given the background knowledge such developers have, that most topics can be translated into books that are between 50 and 100 pages.

This is exactly what we resolved to accomplish with the *Succinctly* series. Isn't everything wonderful born out of a deep desire to change things for the better?

The best authors, the best content

Each author was carefully chosen from a pool of talented experts who shared our vision. The book you now hold in your hands, and the others available in this series, are a result of the authors' tireless work. You will find original content that is guaranteed to get you up and running in about the time it takes to drink a few cups of coffee.

Free forever

Syncfusion will be working to produce books on several topics. The books will always be free. Any updates we publish will also be free.

Free? What is the catch?

There is no catch here. Syncfusion has a vested interest in this effort.

As a component vendor, our unique claim has always been that we offer deeper and broader frameworks than anyone else on the market. Developer education greatly helps us market and sell against competing vendors who promise to “enable AJAX support with one click,” or “turn the moon to cheese!”

Let us know what you think

If you have any topics of interest, thoughts, or feedback, please feel free to send them to us at succinctly-series@syncfusion.com.

We sincerely hope you enjoy reading this book and that it helps you better understand the topic of study. Thank you for reading.

Please follow us on Twitter and “Like” us on Facebook to help us spread the word about the *Succinctly* series!

About the Author

Parikshit Savjani is a Microsoft Certified Solution Expert and Microsoft Certified Trainer working as a Premier Field Engineer with Microsoft specializing in SQL Server and business intelligence (SSAS, SSIS, and SSRS). His role involves consulting, educating, mentoring, and supporting the premier customers of Microsoft. He has more than six years of experience with Microsoft, during which he has authored and developed a number of intellectual properties (IPs) in the SQL and business intelligence space.

While supporting and consulting for premier customers of Microsoft, he has gained experience in working in varied complex environments, understanding common customer bottlenecks, and how to overcome them.

He contributes to the community as well by blogging at sqlserverfaq.net and MSDN Blogs, and by delivering community sessions in SQL Virtual PASS and SQL Server Days.

Chapter 1 Introduction to SQL on Azure

In recent years, we have witnessed how information technology has contributed to the growth and reach of businesses around the globe. Companies which are primarily technology companies, like Amazon, Facebook, and Uber, have built their businesses solely on the basis of information technology and have grown unimaginably. In today's world we cannot imagine a business without a website, app, or web presence, but does having a website, app, or web presence guarantee a business will be successful? Until a business is successful, companies aren't sure whether their investment in information technology and IT infrastructure is fruitful or not.

Setting up and maintaining a website, app, database, or data warehouse incurs a lot of capital and operational expenses (CAPEX and OPEX) for a company whose core business may not be IT. As a result, small and medium business around the world are looking towards cloud solution providers to offer IT as a service and pay only as they use with elastic scale up-down flexibility. Even large enterprises who can afford their own data centers and infrastructure are looking towards cloud providers since data is growing at a rapid pace and investing in new hardware and infrastructure for new projects, development, and testing may not make sense until the project becomes critical and revenue generating.

Cloud services allow big enterprises to innovate and research at an optimized cost without investing upfront on the infrastructure in their data centers. Further, when products are designed to run as a service, the product release cycle can be agile and fast. All of these benefits make cloud services an attractive offering and optimal alternative for businesses today.

In this book, I will introduce you to Microsoft SQL Server (RDBMS) Offerings on Azure and talk about the use cases, considerations, configurations, and design optimizations for running SQL Server on Azure to give you predictable performance at optimized cost. To better understand and follow along with the book, you should obtain an Azure Platform subscription. You can purchase an Azure subscription or sign up for an Azure free trial subscription. Prior knowledge of Powershell will make it easier to understand the scripts used in the book. You can use Windows PowerShell to perform a variety of tasks in Azure, either interactively via a command prompt or automatically through scripts. Azure PowerShell is a module that provides cmdlets to manage Azure through Windows PowerShell. You can download and install the Azure PowerShell modules by running the Microsoft Web Platform Installer. The scripts used in the book can be downloaded from the following location: https://bitbucket.org/syncfusiontech/sql-on-azure-succinctly.

This book is primarily focused on instructing individuals with prior knowledge of SQL Server how to manage, optimize, and design applications running SQL Server on Azure. Covering the basics of SQL Server management and administration is outside the scope of this book.

Microsoft Azure Offerings

Azure is Microsoft's cloud computing platform that provides a collection of integrated service offerings, namely analytics, computing, database, mobile, networking, storage, and web. Microsoft is making huge investments in Azure to ensure its cloud offerings provide a similar or even better set of functionality compared to its on-premise or boxed product offerings. Microsoft's Azure offerings are primarily organized into three categories: Infrastructure as a Service (IaaS), Platform as a Service (PaaS), and Software as a Service (SaaS). The following figure best describes the differences between IaaS, PaaS, and SaaS.

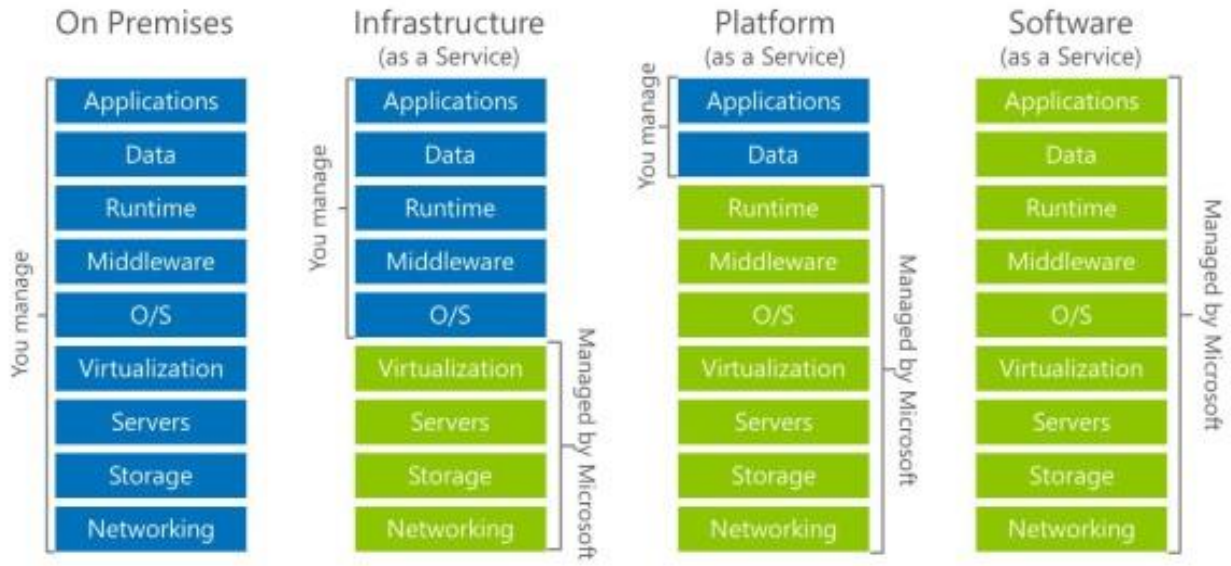

Figure 1: Microsoft's Azure Offerings

SQL Server on Azure is available in two offerings: SQL on Azure VMs (IaaS) and Azure SQL Database (PaaS). Let us examine each of these offerings and how they are different from their on-premise counterparts.

SQL Server on Azure VMs (IaaS)

SQL Server on Azure Virtual Machine (VMs) is no different from SQL Server running locally except for the fact that SQL Server will be running on a VM in a Microsoft data center. Azure VMs support running all supported on-premise versions of SQL Server. This also means the application code, scripts, and monitoring running on on-premise versions of SQL Server are also compatible with and can be easily migrated to Azure VM and vice-versa with minimal hassle and modifications. We will discuss more on considerations for running SQL Server in Azure VM and contrast it with its on-premise counterpart in later sections and in more detail in later chapters.

Azure SQL Database (PaaS)

Azure SQL Database, which is the Database as a Service offering of SQL Server, is a newly designed cloud service in Azure suitable for cloud-designed applications. In this offering, the database is provisioned and hosted on Microsoft data centers, and you do not need to worry about setup, configuration, resource tuning, high availability, or backups for SQL Server. The Azure portal provides the connection string to connect to the SQL Database, which is used by the application/DBAs to create the schema and to insert, manipulate, query, or administer the data. When this service was first introduced, it was still catching up with its on-premise counterpart in terms of functionality and features, but the latest version of the service exceeds SQL 2014 with support for columnstore indexes, row security, query store, and more. Azure SQL Database is better suited for SaaS applications using scale-out patterns. We will discuss use-case scenarios for Azure SQL Database and its design consideration in more detail in following chapters.

The following table compares the SQL Server on Azure VMs (IaaS) versus Azure SQL Database (PaaS) in terms of use-case scenarios, features and cost. It can be viewed on Microsoft's site at the following link: https://azure.microsoft.com/en-us/documentation/articles/data-management-azure-sql-database-and-sql-server-iaas/.

Table 1: Comparing Azure SQL Database (PaaS) and SQL Server in Azure VM (IaaS)

	Azure SQL Database (PaaS)	SQL Server in Azure VM (IaaS)
Best for	New cloud-designed applications that have time constraints in development and marketing. SaaS applications.	Existing Dev Cert and test environment applications that require fast migration to the cloud with minimal changes that do not need on-premise nonproduction SQL Server hardware. SQL Server applications that require accessing on-premise resources (such as Active Directory) from Azure via a secure tunnel. Rapid development and test scenarios for disaster recovery for on-premise SQL Server applications using backup on Azure Storage or AlwaysOn replicas in Azure VMs.
SQL Support	Near SQL-like functionality in preview. Application should be supported on Azure SQL Database. Existing environments cannot be migrated to Azure Database unless the application is rewritten.	All SQL versions (SQL 2005 – SQL 2014) are supported, similar to on-premise functionality. Existing applications can be migrated with minimal coding effort.
Size Constraints	Databases of up to 500 GB in size.	Unlimited database size.
Total cost of ownership	Eliminates hardware costs. Reduces administrative costs.	Eliminates hardware costs.

	Azure SQL Database (PaaS)	SQL Server in Azure VM (IaaS)
Business continuity	In addition to built-in fault tolerance infrastructure capabilities, Azure SQL Database provides features, such as Point in Time Restore, Geo-Restore, and Geo-Replication to increase business continuity. For more information, see Azure SQL Database Business Continuity.	SQL Server in Azure VM lets you to set up a high availability and disaster recovery solution for your database's specific needs. Therefore, you can have a system that is highly optimized for your application. You can test and run failovers by yourself when needed. For more information, see High Availability and Disaster Recovery for SQL Server in Azure Virtual Machines.
Hybrid cloud	Your on-premises application can access data in Azure SQL Database. But from SQL Database you can't access resources on-premise	With SQL Server in Azure VMs, you can have applications that run partly in the cloud and partly on-premises. For example, you can extend the on-premises network and Active Domain Directory to the cloud via Azure Network Services. In addition, you can store on-premises data files in Azure Storage using the SQL Server Data Files in Azure feature. For more information, see Introduction to SQL Server 2016 Hybrid Cloud.

Motivation for Running SQL Server on Azure

With the addition of SQL on Azure Offerings, enterprises today have four options for running SQL Server in their hybrid cloud environments:

- SQL Server on physical machines
- SQL Server on VMWare or Hyper-V VMs
- SQL Server on Azure (IaaS)
- Azure SQL Database (PaaS)

It is important to understand what the merits and demerits are for running SQL Server on Azure for organizations to be able to decide which scenarios are suitable for running SQL Server on Azure versus another solution.

The following figure compares the above options for running SQL Server in an environment in terms of infrastructure and administration cost. It can be viewed on Microsoft's site by using the following link: https://azure.microsoft.com/en-us/documentation/articles/data-management-azure-sql-database-and-sql-server-iaas/.

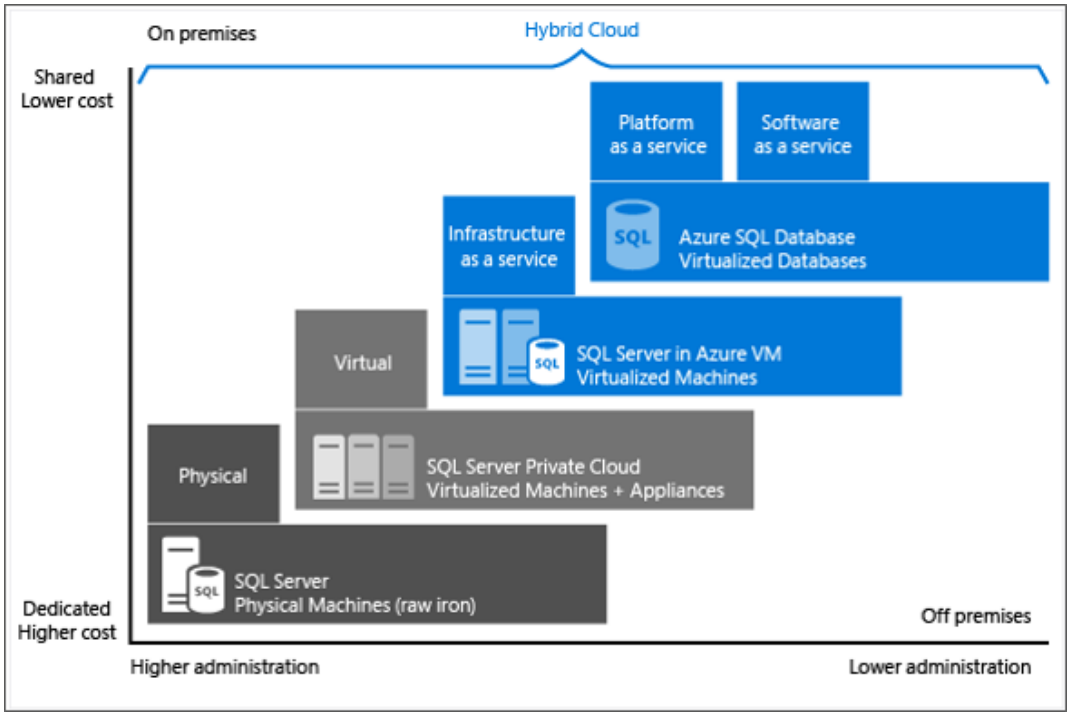

Figure 1: Motivation for Running SQL Server on Azure

As we move from physical to virtual to Azure, the infrastructure and administration cost of SQL Server is reduced. Running in VMs also gives us elastic scale up/down flexibility depending on the user workload, which gives us additional benefits in terms of cost since we pay only for the resources which we consume and can scale down or shut down when they are not in use.

Elastic Scale up/down can be particularly useful in applications that have skewed workloads, such as software for retail stores that see higher sales during Thanksgiving or holiday seasons. Such workloads create higher demand for computing power, and servers need to be scaled up during the applicable time period. Another good example would be test or certify servers, which need to be of equivalent capacity to production servers to provide accurate application behavior, but only need to be active during the testing phase and scaled down or turned off when not in use.

Another benefit of running SQL Server on Azure is lower turnaround time for setting up and configuring SQL Server or databases. The turnaround time is quickest for Azure SQL Database since you do not need to worry about the server, OS, setup, or configuration, and within a few minutes a database is provisioned and available to accept new connections. For SQL Server on Azure VMs there are images available in Azure gallery which automate the entire installation and configuration of SQL Server, allowing for minimal steps and reduced time for spinning a SQL Server instance.

The primary motivations for moving SQL Server workloads to Azure are:

- Low hardware and data center costs.
- Low administration costs.

- Elastic scale.
- Low turnaround time for setup and configuration.

Considerations for Running SQL Server on Azure

In spite of the cost savings and efficiency of running SQL Server on Azure, there are scenarios where it might not be appropriate to run SQL Server on Azure. We will discuss such scenarios in this section.

Data Security

In many organizations, some databases are used for storing highly confidential data or customer personally identifiable information (PII), which the organization may not be comfortable with storing on Azure in Microsoft data centers. Companies with such security concerns might prefer the data to reside in on-premise servers.

Application Affinity

For most applications, it is preferred that the database reside in the same LAN or network vicinity as the application to avoid any network latency-related performance problems. So if the application is designed to run on-premise, you might prefer to run SQL Server on-premise as well to avoid network delays. Large enterprises can set up ExpressRoute connections to Microsoft Azure data centers, but they still incur some network latency which might be significant enough for applications where response time is critical.

Critical Production Workloads

Microsoft Azure is a shared multi-tenant environment where the cost benefit is achieved by sharing and reusing resources. Since it is a shared environment, there are times when you may not see predictable performance if the host servers are over-subscribed or if the storage channels are saturated. If you are critical about the performance of SQL Server and looking for predictable performance, it would be ideal to run SQL Server on-premise in a dedicated environment. This might involve higher cost, but it ensures predictable performance.

Where to Host My SQL Database? (Decision Matrix)

After understanding the motivations and considerations for running SQL Server on Azure, the following decision matrix should help individuals and organizations decide where to host their SQL Server environment.

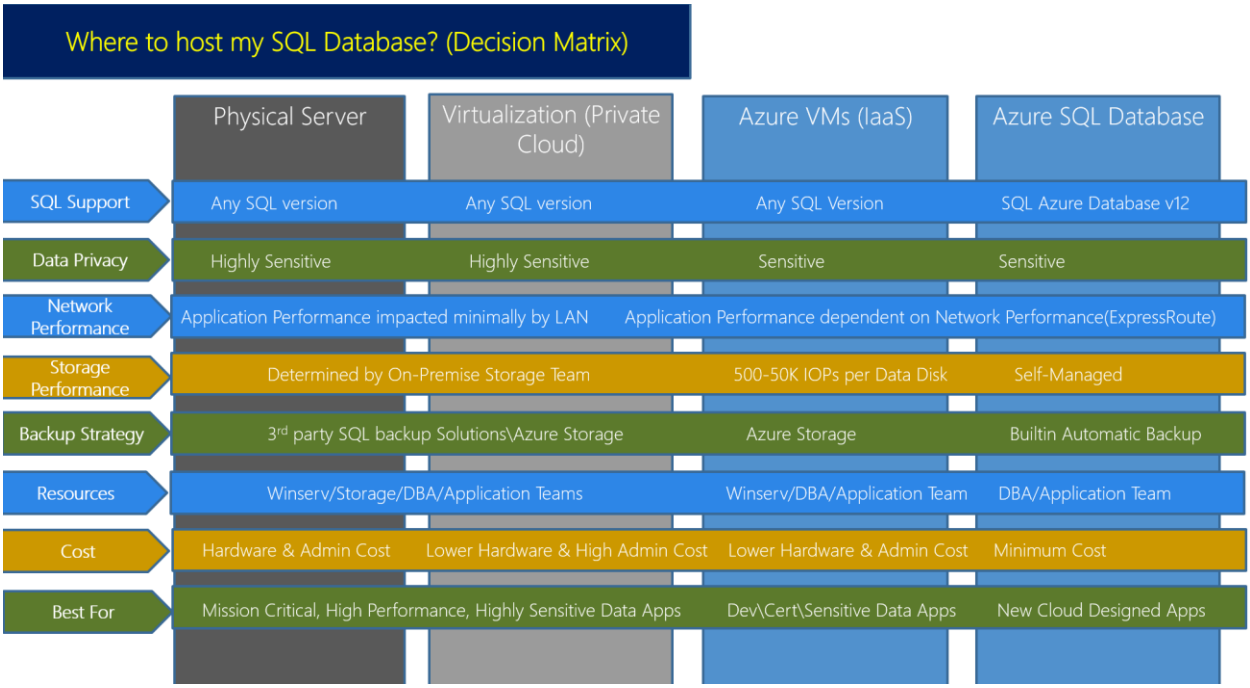

Figure 2: SQL Database Optimal Environment Decision Matrix

This chapter introduced you to the SQL Server offerings in Azure and discussed the motivations and considerations for moving your SQL workload to Azure. In the following chapters, we will go into detail on each of these offerings and explore how to design applications and configure SQL Server in Azure to gain maximum benefit at optimized cost.

Chapter 2 Installing and Configuring SQL Server on Azure VMs

To create a SQL Server virtual machine in Azure, you must first obtain an Azure Platform subscription. You can purchase an Azure subscription or sign up for an Azure free trial subscription.

Microsoft Azure supports the following options to install and configure SQL Server on Azure Virtual Machines:

- Create a Syspreped image of SQL Server in Hyper-V and upload it to Azure.
- Provision a Windows image from Azure Gallery and install SQL Server on it.
- Provision a SQL Server image from Azure Gallery using the portal or PowerShell.

The method you choose is largely influenced by the SQL Server licensing policy for your deployment. With SQL Server in Azure VM, you can either use the platform-provided SQL Server image or bring your SQL Server license to Azure. When using the SQL Server platform-provided images, the cost depends on the VM size as well as the version of SQL Server you choose. Basically, you pay the per-minute licensing cost of SQL Server, the per-minute licensing cost of Windows Server, and the Azure storage cost. The per-minute billing option allows you to use SQL Server for as long as you need it without buying a full SQL Server license. If you bring your own SQL Server license to Azure, you are charged for Azure compute and storage costs only. For more information, see License Mobility through Software Assurance on Azure.

In this chapter, we will briefly discuss the first two options and focus mainly on provisioning and configuring a SQL Server image from Azure Gallery using PowerShell cmdlets. In a later part of this chapter, we will discuss post-deployment tasks that need to be performed after SQL Server is installed.

Creating a Sysprepped Image of SQL Server in Hyper-V and Adding it to Azure

This method is used for installing SQL on Azure VM when you would prefer to bring your own license, as supported by License Mobility for Azure, and would like to pay only for Azure compute and storage costs incurred for hosting your VM on Azure. SQL Server supports the preparation of SysPrep images for installation starting with SQL 2008 R2; hence, it is easy to create a sysprepped image of SQL Server.

This method involves the following steps:

1. Prepare a SysPrep image with SQL Server installed in a Hyper-V VM.

2. Upload the VHD file of the VM to Azure Blob storage.

3. Create an image from the uploaded VHD from Azure Management portal or PowerShell.

Note: The VHDX format is not supported in Microsoft Azure. You can convert the disk to VHD format using Hyper-V Manager or Convert-VHD PowerShell cmdlet.

The following articles provide detailed instructions on creating and uploading a Windows Server VHD to Azure.

https://azure.microsoft.com/en-us/documentation/articles/virtual-machines-create-upload-vhd-windows-server/

Provision a Windows Gallery Image and Install SQL Server on the VM

This method is used for installing SQL on Azure VM when you want to bring your SQL Server license but not your Windows license. In this case, you pay per-minute for the Azure Compute, Storage, and Windows license but not for the SQL Server license.

This method involves the following steps:

1. Provision a Windows Server image from Azure Gallery as shown in the following image.

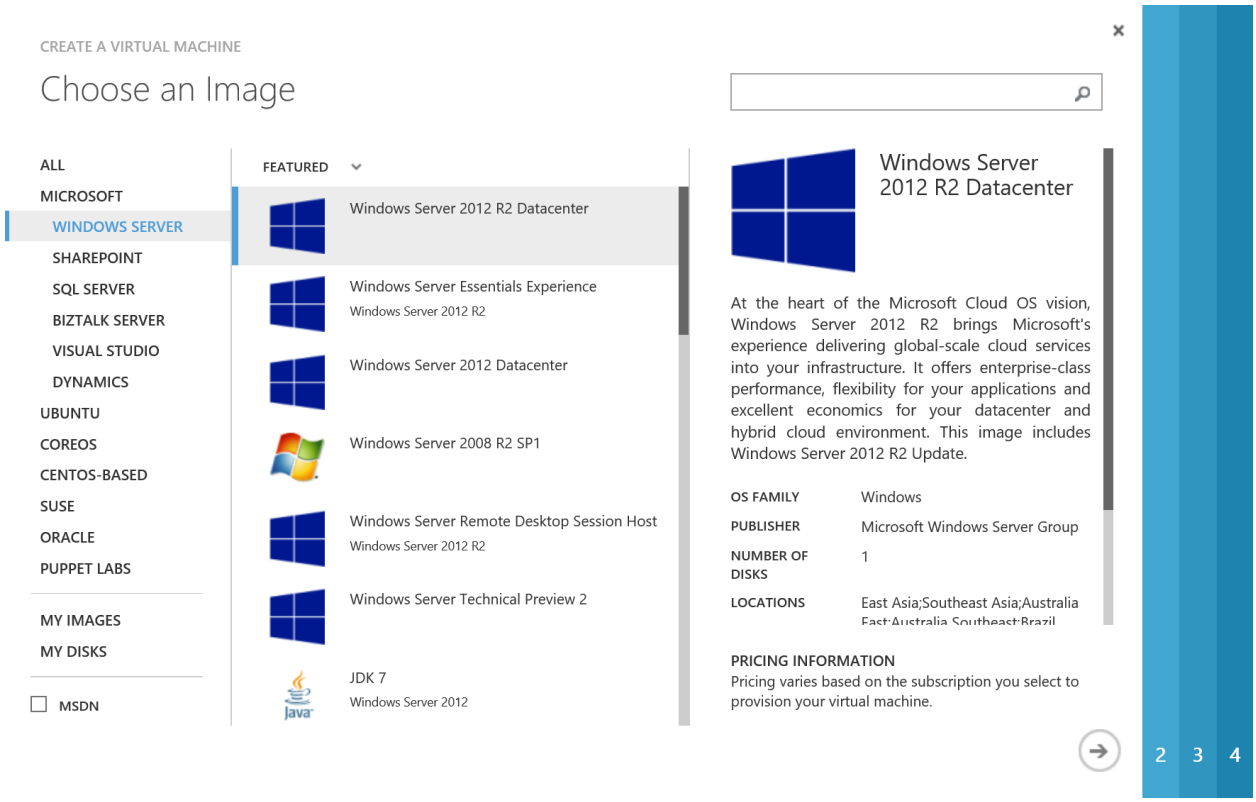

Figure 3: Provisioning a Windows Server Image from Azure Gallery

2. Download or copy the SQL Server setup files into the VM.

3. Install SQL Server on the VM using your own license's PID.

The following article provides detailed instructions on provisioning a Windows Server image from Azure Gallery:

https://azure.microsoft.com/en-us/documentation/articles/virtual-machines-windows-tutorial/

Provisioning a SQL Server image from Azure Gallery using the Azure Management Portal

This method is used when you do not own or do not want to use your SQL Server license and would prefer to pay per-minute for a SQL Server license along with an Azure Compute, Storage, and Windows license. This is best for short-term projects or if you are unsure if the current project will be sustainable and would like to start with an Azure platform-provided SQL Server image.

This method involves provisioning a VM from the list of pre-configured SQL Server images available in Azure Gallery. The Azure Gallery provides all the SQL Server build images from SQL 2008 R2-SQL 2016 at various Service Pack levels, which greatly reduces the time to build, install, and configure SQL Server at the desired Service Pack level.

The following figure gives a glimpse of SQL Server images in Azure Gallery. The figure doesn't show the complete list of available SQL Server images since the list is too long to be covered in a single frame.

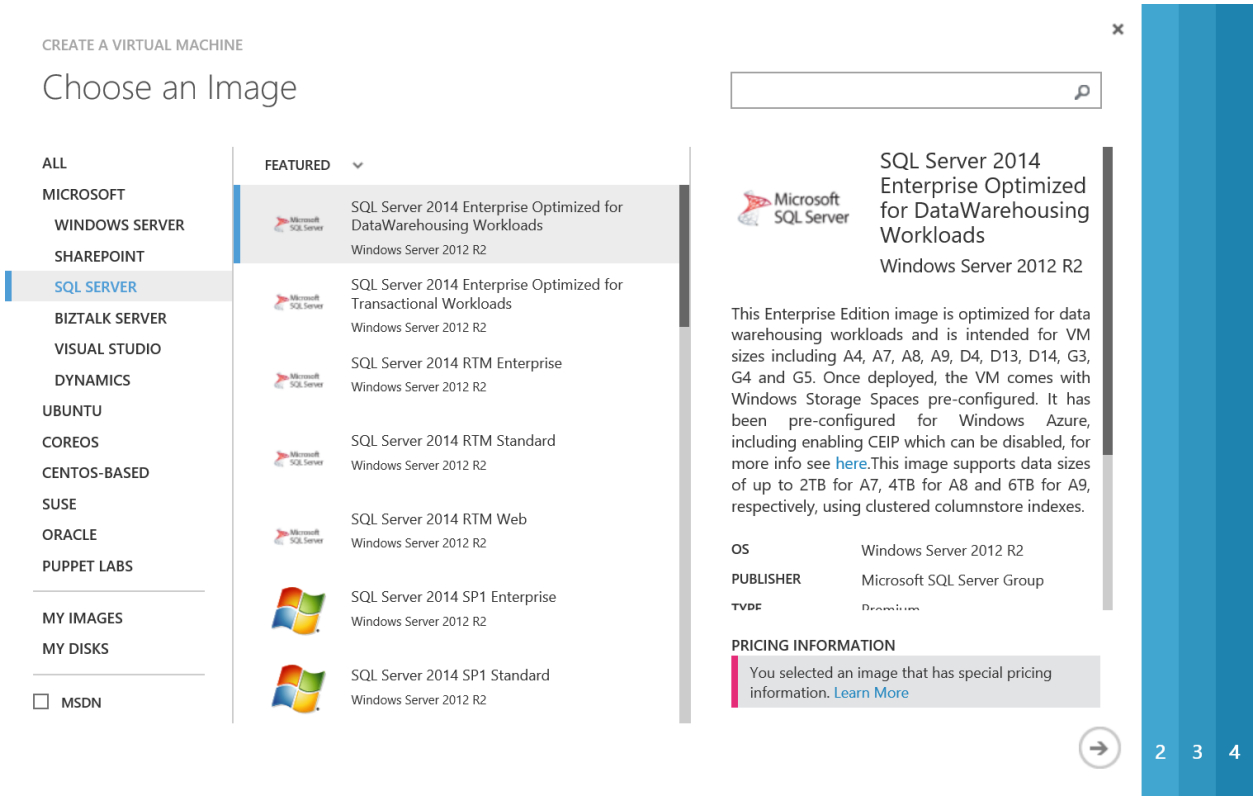

Figure 4: SQL Server Images Available in Azure Gallery

The detailed steps to provision a SQL Server image using the portal are described in the following article:

https://azure.microsoft.com/en-us/documentation/articles/virtual-machines-provision-sql-server/

Provisioning a SQL Server Image from Azure Gallery Using Azure PowerShell

This method is no different from the previous one except that the VM is provisioned using Azure PowerShell. You can use Windows PowerShell to perform a variety of tasks in Azure, either interactively at a command prompt or automatically through scripts. Azure PowerShell is a module that provides cmdlets to manage Azure through Windows PowerShell. You can use the cmdlets to create, test, deploy, and manage solutions and services delivered through the Azure platform. In most cases, you can use the cmdlets to perform the same tasks that you can perform through the Azure Management Portal. For example, you can create and configure cloud services, virtual machines, virtual networks, and web apps.

You can download and install Azure PowerShell modules by running the Microsoft Web Platform Installer. When prompted, click Run. The Web Platform Installer installs the Azure PowerShell modules and all dependencies. Follow the prompts to complete the installation. This installation will also create an Azure PowerShell console, which you can access from your Programs or Apps list, depending on the version of Windows you are using.

Note: In this section we will discuss only the PowerShell cmdlets relevant for the provisioning of a SQL Server image in Azure. The complete list of Azure PowerShell cmdlets, along with their definitions, can be found in the following link: https://msdn.microsoft.com/library/azure/jj554330.aspx

Connecting to an Azure Account

In order to connect to an Azure account using PowerShell, use the following steps:

1. Open the Azure PowerShell console.
2. Type the following command in the PowerShell window.

Code Listing 1: Connecting to an Azure Account

```
Add-AzureAccount
```

3. In the window, type the email address and password associated with your account.
4. Azure authenticates and saves the credentials and then closes the window.

Selecting an Azure Subscription

An Azure account can be mapped to multiple Azure subscriptions. Use the following cmdlet to enlist all the Azure subscriptions mapped to your Azure account.

Code Listing 2: Identify the List of Azure Subscriptions Mapped to the Azure Account

```
Get-AzureSubscription
```

If there are multiple subscriptions assigned, it is important to choose the subscription to which you would like to provision the VM since each subscription might be mapped to different Azure locations and support different VM sizes.

Use the following cmdlet to select the desired subscription. The subscription name can be obtained from the output of the *Get-AzureSubscription* cmdlet

Code Listing 3: Selecting the Azure Subscription to Be Used for Provisioning a VM

```
Select-AzureSubscription –SubscriptionName <<Subscription Name>>
```

Creating or Identifying an Azure Storage Account

Before preparing to provision a VM in Azure, you need to have an Azure Storage account which will be used to host the VHDs for the VMs. A given Azure subscription can have multiple storage accounts. Every storage account has a limitation on the number of IOPs it can handle, so it is recommended that multiple Storage accounts be created to avoid storage contention.

To identify the list of Azure Storage accounts created in the Azure Subscription, use the following cmdlet:

Code Listing 4: Identify the List of Azure Storage Accounts Available to the Current Subscription

```
Get-AzureStorageAccount
```

To create a new Azure Storage account in the given Subscription, use the following cmdlet:

Code Listing 5: Creating a New Azure Storage Account in an Azure Subscription

```
New-AzureStorageAccount -StorageAccountName <<String>> -AffinityGroup
<<String>> -Description <<String>> -Label <<String>> -Type <<String>>

New-AzureStorageAccount [-StorageAccountName] <<String>> -Location
<<String>> -Description <<String>> -Label <<String>> -Type <<String>>
```

The AffinityGroup or Location parameters are mutually exclusive and cannot be used together. AffinityGroup specifies the name of an existing affinity group in the current subscription, while Location specifies the location of the Azure data center where the storage account was created. You can get the locations supported by the given subscription by firing the *Get-AzureLocation* cmdlet.

The Type parameter specifies the type of the storage account. Valid values are:

- Standard_LRS
- Standard_ZRS
- Standard_GRS
- Standard_RAGRS

If you do not specify this parameter, the cmdlet uses a default value of Standard_GRS. Standard_ZRS accounts cannot be changed to other account types, and vice versa.

Creating a New Azure Cloud Service

Cloud Service is a DNS name for the Azure connectivity service. In order to access the VM remotely, you need to ensure it is mapped to Cloud Service. Further, Cloud Service must be created in the same location as the storage account. The following PowerShell cmdlet is used to create a new Cloud Service:

Code Listing 6: Creating a New Cloud Service for Provisioning of VM

```
New-AzureService –ServiceName <<New ServiceName>> –Location
<<Locationname>>
```

Identifying the SQL Server Image To Be Provisioned

As discussed earlier, Azure Gallery provides a list of SQL Server images ranging from SQL 2008 R2–SQL 2016 at various Service Pack levels. The following PowerShell cmdlet can be used to list and filter the images for SQL 2014 images.

Code Listing 7: Getting a List of SQL 2014 Images from the Gallery

```
Get-AzureVMImage | where { $_.os –eq “Windows” –and $_.ImageFamily –like
“*SQL*2014*” } | Sort-Object -Property PublishedDate -Descending
```

Identifying the Azure VM Size To Be Provisioned

The VM size decides the compute capacity as well the per-minute cost of the VM. The virtual machines in Azure are categorized as A-series, D-series, DS-series, and G-series VMs. You should understand the difference between each of them first.

A-series VMs have basic Azure data disks that can support up to 500 IOPs at moderate latency. As you move from sizes A0 to A11 the number of cores and amount of memory increases. Furthermore, the VM sizes A8-A11 use Intel® Xeon® E5-2670 2.60 Ghz, which provides superior computing performance compared to lower VM sizes. As you move from A0 to A11, the per-minute compute cost of the VM also increases proportionally.

D-series VMs were introduced later to provide fast performance for applications needing quick, local ephemeral storage. D-series VMs are similar to A-series VMs, except for the fact that they have a local temporary storage drive (D drive), which is solid-state and provides fast, low-latency response. In the case of SQL Server, the temporary drive is useful for hosting the tempdb file, which is cleaned and recreated on every restart of the service.

DS-Series VMs are new Azure premium storage-backed VMs. Azure Premium Storage is a new type of SSD-based storage, designed to support I/O intensive workloads. With Premium Storage, you can provision a persistent disk and configure the size and performance characteristics that will meet your requirements. You can then attach several Premium Storage-backed data disks to a VM, stripe across them and deliver to your applications up to 32 TB of storage per VM with more than 50,000 IOPS per VM with extremely low latencies. For more information on Premium Storage, see Premium Storage Overview.

G-series VMs are the most recent addition to the family. They provide the most memory, the highest processing power, and the largest amount of local SSD of any VM size currently available in the public cloud. G-series offers up to 32 vCPUs using the latest Intel® Xeon® processor E5 v3 family, 448GB of memory, and 6.59 TB of local Solid State Drive (SSD) space. This powerful VM size easily handles deployments of mission critical applications such as large relational database servers (SQL Server, MySQL, etc.) and large NoSQL databases (MongoDB, Cloudera, Cassandra, etc.).

Full detailed specs of each VM size offered in Azure are available in the following article: https://azure.microsoft.com/en-us/documentation/articles/virtual-machines-size-specs/

Use the following PowerShell cmdlet to identify the VM sizes supported by the current Azure subscriptions. Not all VM sizes are available in all Azure locations.

Code Listing 8: Listing the VM Role Size Available in Given Azure Subscription

```
Get-AzureRoleSize
```

Provisioning a SQL Server Image VM

The following cmdlet is used to provision a new Azure VM of specified size within the specified Cloud Service with the specified image from Azure Gallery:

Code Listing 9: Provisioning a SQL Server Image VM

```
New-AzureQuickVM -Windows -ServiceName <<ServiceName>> -Name <<VMName>> -
ImageName <<ImageName.imagename>> -Password <<password>> -AdminUsername
<<AdminAccount>> -InstanceSize <<VmSize>> -EnableWinRMHttp | out-null
```

The *–EnableWinRMHttp* command is used to open up the endpoints for Remote Desktop and PowerShell remoting.

Finally, using all the cmdlets discussed above, we create the following script, which can be used to automate the SQL Server VM provisioning in Azure:

Code Listing 10: Complete Script to Automate Provisioning of SQL Server Image VM in Azure

```
# The original script for Provisioning of SQL Server Azure VM was created
by Sourabh Agrawal and Amit Banerjee
```

```
# The script is modified by Parikshit Savjani to suit the readers of the
book SQL on Azure Succintly

Add-AzureAccount | out-null

# Make sure the Authentication Succeeded.

If ($?)

{

      Write-Host "`tSuccess"

}

Else

{

      Write-Host "`tFailed authentication" -ForegroundColor Red

      Exit

}


#Select the desired Azure Subscription

[array] $AllSubs = Get-AzureSubscription

If ($AllSubs)

{

        Write-Host "`tSuccess"

}

Else

{

        Write-Host "`tNo subscriptions found. Exiting." -ForegroundColor
Red

        Exit

}

#Write-Host "`n[Select Option] - Select a Subscription to Work With" -
ForegroundColor Yellow

$count = 1

ForEach ($Sub in $AllSubs)
```

```
{
   $SubName = $Sub.SubscriptionName
   Write-host "`n$count - $SubName"  -ForegroundColor Green
   $count = $count+1
}
$SubscriptionNumber = Read-Host "`n[SELECTION] - Select a Subscription to
Provision the VM"
If($SubscriptionNumber -gt $count)
{
    Write-Host "`Invalid Subscription Entry - Existing" -ForegroundColor
Red
    Exit
}
$SelectedSub = Get-AzureSubscription -SubscriptionName
$AllSubs[$SubscriptionNumber - 1].SubscriptionName 3>$null
$SubName = $SelectedSub.SubscriptionName
Write-Host "`n Selected Subscription - $SubName" -ForegroundColor Green
$SelectedSub | Select-AzureSubscription | out-Null


#Identify the Azure Storage Account to be used to provision the VM
$StorAccName = Read-Host "`nEnter Azure Storage Account Name"
if(Get-AzureStorageAccount -StorageAccountName $StorAccName)
{
    Write-Host "`n[INFO] - Using Storage Account - $StorAccName" -
ForegroundColor Yellow
}
else
{
    Write-Host "Storage Account $StorAccName does not Exists - Please
Create it" -ForegroundColor Red
    Exit
```

```
}


# Identify the Cloud Service to use to Provision the VM

$ServiceName = Read-Host "`nEnter your Cloud Service Name for VM"

if(Get-AzureService -ServiceName $ServiceName)

{

Write-Host "`n [INFO] - Cloud Service $ServiceName already exists, using
the same..." -ForegroundColor Yellow

}

else

{

$Location = (Get-AzureStorageAccount -StorageAccountName
$StorAccName).Location.ToString() 3>$null

New-AzureService -ServiceName $ServiceName -Location $Location|Out-Null

Write-Host "`n [INFO] - Cloud Service $ServiceName created..." -
ForegroundColor Yellow

}

$azureService = Get-AzureService -ServiceName $ServiceName


#Get the name for the Azure VM

$VMName = Read-Host "`n Enter the name for the Azure VM"

$VMName = $VMName.ToLower()


#Using PreConfigured Values for VM Size

$VmSize = "Standard_D11"


[array] $AllImageFamily = get-azurevmimage |where {$_.os -eq "Windows" -and
$_.ImageFamily -like "*SQL*2014*" } | Sort-Object -Property PublishedDate -
Descending

$cnt = 1

ForEach ($ImgFamily in $AllImageFamily)
```

```
{
   $ImgName = $ImgFamily.Label
   Write-host "`n$cnt - $ImgName"  -ForegroundColor Green
   $cnt = $cnt+1
}
$ImageNo = Read-Host "`n[SELECTION] - Select the Image to install"
If($ImageNo -gt $cnt)
{
    Write-Host "`Invalid Subscription Entry - Existing" -ForegroundColor
Red
    Exit
}


$ImgName =$AllImageFamily[$ImageNo - 1].Label


Write-Host "`nCreating a VM of size $VMSize, with $ImgName " -
ForegroundColor Green


#Identify the Admin Account and Password to be used for VM
$AdminAccount = Read-Host "`n Enter the Admin account name"
$password = Read-Host "`n Enter the password"


#Select the Storage to be used and create the VM.
Set-AzureSubscription -SubscriptionName $SubName -CurrentStorageAccountName
$StorAccName


Write-Host "`n[INFO] - Script is creating a $ImgName image, Please
wait...." -ForegroundColor Yellow
New-AzureQuickVM -Windows -ServiceName $ServiceName -Name $VMName -
ImageName $AllImageFamily[$ImageNo - 1].imagename -Password $password -
AdminUsername $AdminAccount -InstanceSize $VmSize -EnableWinRMHttp | out-
null
```

```
#Check to make sure that VM was created
$CreatedVM = Get-AzureVM -ServiceName $ServiceName -Name $VMName -
ErrorAction SilentlyContinue
If ($CreatedVM)
{
      Write-Host "`tVM Created Successfully"
}
Else
{
      Write-Host "`tFailed to create VM" -ForegroundColor Red
      Exit
}
```

Post Deployment Configuration of Provisioned SQL Server VM

Attaching Data Disks

After the creation of SQL Server on Azure VM, it is essential to add a data disk to the VMs that will be used to host the database files. As a best practice for SQL Server, it is recommended to have separate dedicated disks for system databases, transaction log files, user data files, and tempdb files.

In general, we would need to add at least four data disks to our VMs. The maximum data disks that can be attached to a VM is dependent on the VM Size. The max number of data disks supported by A-series (A7-A11) VMs is 16, while that of D-series (D14) VMs is 32 and that of G-series VMs (G5) is 64. Each data disk can have a maximum size of 1024GB (or 1 TB).

We can use the following PowerShell cmdlet to attach a data disk to a VM:

Code Listing 11: Attaching a Data Disk to a VM

```
Add-AzureDataDisk –VM <<vm>> -CreateNew -DiskSizeInGB <<size>> -DiskLabel
<<diskname>> -LUN <<LUN>> -HostCaching None
```

We can use the following PowerShell script to attach four data disks to the SQL Server VM we created earlier:

Code Listing 12: Attaching Multiple Data Disks for SQL Server VM

```
# The original script for Provisioning of SQL Server Azure VM was created
by Sourabh Agrawal and Amit Banerjee

# The script is modified by Parikshit Savjani to suit the readers of the
book SQL on Azure Succintly

#Create and Attach the Disks

Write-Host "`n[INFO] - Script is creating and attaching 4  data disks,
Please wait." -ForegroundColor Yellow

for($i = 0; $i -lt 4; $i++)

{

    $disklabel = "disk$VMName"

    $diskname = $disklabel + $i.ToString()

    Add-AzureDataDisk -VM $vm -CreateNew -DiskSizeInGB 20 -DiskLabel
$diskname -LUN $i -HostCaching None | out-null

}

$vm | Update-AzureVM | out-null


#Make sure all disks are attached and working

$vm = Get-AzureVM -Name $VMName -ServiceName $ServiceName

[array]$CreatedDataDisks = Get-AzureDataDisk -VM $vm

$CreatedDataDisksCount = $CreatedDataDisks.Count


If($CreatedDataDisksCount -eq 4)

{

    Write-Host "`nDisk Creation Successfull - All 4 disks are up and
running" -ForegroundColor Green

}

else

{
```

```
    Write-Host "`nData disk creation failed " -ForegroundColor Red
    Exit
}


#Check to make sure VM is done building
Write-Host "`n[INFO] - Script is checking to see if VM is ready, Please
wait." -ForegroundColor Yellow
$iteration = 0
while($vm.status -ne "ReadyRole" -and $iteration -lt 60)
{
Start-Sleep -Seconds 30
$vm = Get-AzureVM -Name $vmname -ServiceName $ServiceName
write-host "`tVM status is:" $vm.status
$iteration++
}
```

Adding TCP/IP Endpoints for SQL Server Listener

Azure VMs run within the Cloud Service in the Microsoft data center. In order to communicate or connect to the server remotely, you need to create endpoints in the VMs that will open up the firewall ports for communication into the VM for the specified communication protocol.

A default instance of SQL Server listens on TCP port 1433, while a named instance listens on a dynamic port that requires SQL Browser Service running for discovery in order to assist the clients in connecting to SQL Server at the current port on which it is running. However, for a SQL Server instance running on Azure VM, you need to ensure SQL Server runs on a static port to create TCP/IP endpoints only for the ports for which SQL Server is listening. The following blog post describes the necessary steps to assign a static port to a named instance of SQL Server:

http://blogs.msdn.com/b/arvindsh/archive/2012/09/08/how-to-assign-a-static-port-to-a-sql-server-named-instance-and-avoid-a-common-pitfall.aspx

Use the following PowerShell cmdlet to add endpoints to the Azure VMs.

Code Listing 13: Adding an Endpoint to Azure VM

```
Add-AzureEndpoint -Name <<Endpoint Name>> -Protocol tcp -LocalPort
<<LocalPort>> -PublicPort <<Public Port>>  -VM <<vm>>
```

PublicPort is the port which will be used along with the Cloud Service Name by the external clients to connect to the Azure service, while *LocalPort* is the actual port within the VM on which the service is running. The connections to *PublicPort* are internally mapped and routed to *LocalPort* in Azure VM. You can use same port number for *PublicPort* and *LocalPort,* but endpoints also give us flexibility to hide *LocalPort* by keeping the Public and Local port separate.

For the SQL Server VM created earlier, you can use the following PowerShell script to create a SQL listener endpoint with *LocalPort* 1433 and *PublicPort* 14481.

Code Listing 14: Adding a SQL Listener Endpoint to Azure VM

```
Add-AzureEndpoint -Name "SQL Listener Port" -Protocol tcp -LocalPort 1433 -
PublicPort 14481  -VM $vm | Update-AzureVM | Out-Null
```

Connecting to a SQL Server Instance on an Azure VM Using SSMS

Finally, after the installation and configuration of SQL Server on Azure VM, it is important to validate the connectivity to SQL Server from an external client before the system is handed over to the application team.

SQL Server Management Studio (SSMS) is a popular client tool used by DBAs to manage and administer SQL Server instances. You can use SSMS installed on any client workstation to connect to SQL Server. The connection string for your SQL Server would be in the following form:

<Cloud Service Name>.cloudapp.net,<Public Port>

If Azure Virtual Network is configured in your organization, however, the Azure VM can be brought into your network and domain. When the Azure VM is brought into your OU in the active directory, the AD admin can apply the group policy settings which are applicable to any servers in your network. If the Azure VM is brought into your organization's domain, the connection string will be in the following format:

<Cloud Service Name>.<FQDN>,<Public Port>

Once the connectivity to SQL Server instance is verified, SQL Server is ready for application deployment and user workload.

Chapter 3 Migration to SQL Server on Azure VM

As you adopt and embrace the cloud platform, some of the on-premise workload needs to be migrated to Azure for the benefit of cost and efficiency. Usually the Dev/Cert/QA/test environments are the best candidates for initial migration, since these servers have minimum business impact and need not be running 24/7. Migrating these environments also gives maximum cost benefits by running the VMs only during business hours and turning them off when not in use.

In this chapter, we will discuss the various methods for migrating on-premise SQL Server databases to SQL Server on Azure VMs.

Choosing the Right Method for Migration

The primary migration methods are:

- Use the Deploy a SQL Server Database to a Microsoft Azure VM wizard.
- Perform on-premise backup using compression and manually copy the backup file into the Azure virtual machine.
- Perform a backup to URL and restore into the Azure virtual machine from the URL.
- Detach and then copy the data and log files to Azure blob storage and then attach to SQL Server in Azure VM from URL.
- Convert on-premise physical machine to Hyper-V VHD, upload to Azure Blob storage, and then deploy as new VM using uploaded VHD.
- Ship hard drive using Windows Import/Export Service.
- Use Always On to add a replica in Azure and failover to migrate.

The method used for migration is primarily dependent on the target version of the SQL Server instance in Azure VM and the size of the database backup. The following table lists each of the primary migration methods and discusses when the use of each method is most appropriate.

Table 1: Choosing the right method for migration

Method	Source database version	Destination database version	Source database backup size constraint	Notes
Use the Deploy a SQL Server Database to a Microsoft Azure VM wizard	SQL Server 2005 or greater	SQL Server 2014 or greater	> 1 TB	Fastest and simplest method. Use whenever possible to migrate to a new or existing SQL Server instance in an Azure virtual machine.
Perform on-premises backup using compression and manually copy the backup file into the Azure virtual machine	SQL Server 2005 or greater	SQL Server 2005 or greater	Azure VM storage limit	Use only when you cannot use the wizard, such as when the destination database version is less than SQL Server 2012 SP1 CU2 or the database backup size is larger than 1 TB (12.8 TB with SQL Server 2016)
Perform a backup to URL and restore into the Azure virtual machine from the URL	SQL Server 2012 SP1 CU2 or greater	SQL Server 2012 SP1 CU2 or greater	> 1 TB (For SQL Server 2016, < 12.8 TB)	Generally using backup to URL is equivalent in performance to using the wizard and not quite as easy
Detach and then copy the data and log files to Azure blob storage and then attach to SQL Server in Azure virtual machine from URL	SQL Server 2005 or greater	SQL Server 2014 or greater	Azure VM storage limit	Use when attaching database files to SQL Server in an Azure VM storing these files using the Azure Blob storage service, particularly with very large databases

Method	Source database version	Destination database version	Source database backup size constraint	Notes
Convert on-premises machine to Hyper-V VHDs, upload to Azure Blob storage, and then deploy a new virtual machine using uploaded VHD	SQL Server 2005 or greater	SQL Server 2005 or greater	Azure VM storage limit	Use when bringing your own SQL Server license, when migrating a database that you will run on an older version of SQL Server, or when migrating system and user databases together as part of the migration of database dependent on other user databases and/or system databases.
Ship hard drive using Windows Import/Export Service	SQL Server 2005 or greater	SQL Server 2005 or greater	Azure VM storage limit	Use the Windows Import/Export Service when manual copy method is too slow, such as with very large databases

The above table and the methods for migration are taken from Carl Rebeler's article:

https://azure.microsoft.com/en-us/documentation/articles/virtual-machines-migrate-onpremises-database/#ship-hard-drive

Use the Deploy a SQL Server Database to a Microsoft Azure VM Wizard

This wizard available in SQL Server Management Studio (SSMS) is the recommended method for migrating an on-premise user database running on SQL Server 2005 or greater to SQL Server 2014 or greater when the compressed database backup file is less than 1 TB.

Before using the wizard, be sure to download and install the latest version of SSMS on the client workstation. The latest version of this wizard incorporates the most recent updates to the Azure portal and supports the newest Azure VM images in the gallery (older versions of the wizard may not work).

This method requires us to configure an open endpoint for the SQL Server Cloud Adapter service on the Microsoft Azure gateway with a private port of 11435. This port is created as part of SQL Server 2014 or SQL Server 2016 provisioning on a Microsoft Azure VM. The Cloud Adapter also creates a Windows Firewall rule to allow its incoming TCP connections at the default port of 11435. This endpoint enables the wizard to utilize the Cloud Adaptor service to copy the backup files from the on-premise instance to the Azure VM. For more information, see Cloud Adapter for SQL Server.

You can use the following Azure PowerShell cmdlet to add a SQL Server Cloud Adapter Endpoint on port 11435.

Code Listing 15: Adding a SQL Server Cloud Adapter Endpoint to Azure VM

```
Add-AzureEndpoint -Name "SQL Server Cloud Adapter" -Protocol tcp -LocalPort
11435 -PublicPort 11435  -VM $vm | Update-AzureVM | Out-Null
```

This method involves the following steps:

1. Open Microsoft SQL Server Management Studio for Microsoft SQL Server 2016 and connect to the SQL Server instance containing the user database that you will be migrating to an Azure VM.

2. Right-click the database that you are migrating, select Tasks, and then click Deploy Database to a Microsoft Azure VM as shown in the following figure:

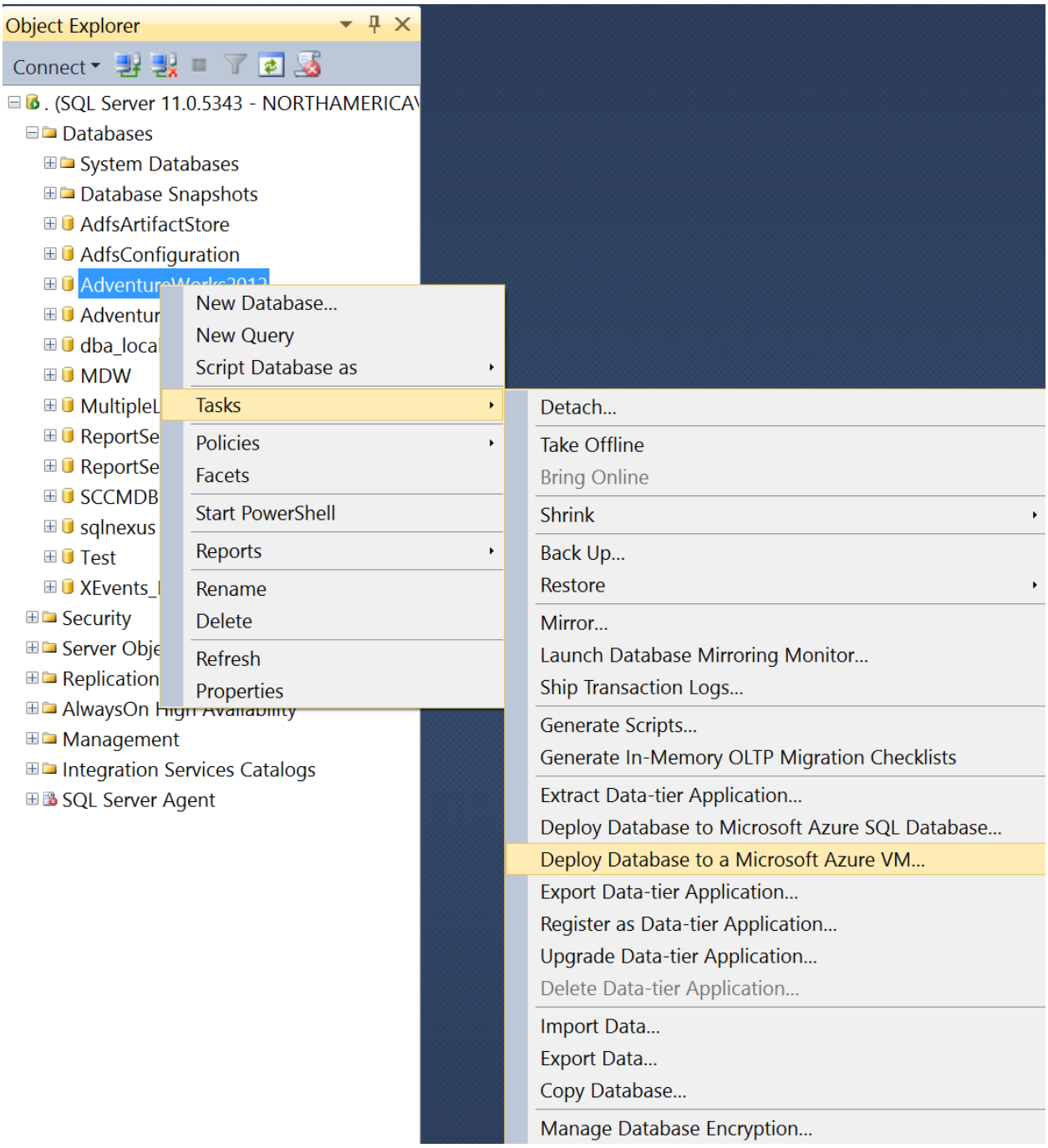

Figure 5: Deploy Database to Microsoft Azure VM Wizard

3. On the introduction page, click Next. On the Source Settings page, connect to the SQL Server instance containing the database to migrate.

4. On the Source Setting page, specify a temporary location to hold the backup files that is read-write accessible to the source SQL Server instance and click Next.

5. On the Microsoft Azure sign-in page, sign-in to your Azure account and select the subscription you want to use.

6. On the Deployment Settings page, you can specify a new or existing Cloud Service name and virtual machine name:

    - Specify a new Cloud Service name and virtual machine name to create a new Cloud Service with a new Azure virtual machine using a SQL Server 2014 or SQL Server 2016 Gallery image.

        - If you specify a new Cloud Service name, specify the storage account that you will use.

        - If you specify an existing Cloud Service name, the storage account will be retrieved and entered for you.

    - Specify an existing Cloud Service name and new Virtual Machine name to create a new Azure virtual machine in an existing Cloud Service. Only specify a SQL Server 2014 or SQL Server 2016 gallery image.

    - Specify an existing Cloud Service name and Virtual Machine name to use an existing Azure virtual machine. This must be an image built using a SQL Server 2014 or SQL Server 2016 gallery image.

7. Click Settings on the Deployment Settings page.

    If you specified an existing Cloud Service and VM, you will be prompted for your admin username and password to connect to the VM.

    If you specified a new virtual machine, you will be prompted to enter the following information:

    - Image (Select SQL 2014 or SQL 2016 Images from the Azure Gallery)

    - Username

    - New password

    - Confirm password

    - Location

    - Size

    In addition, you need to check the box to accept the creation of new, self-generated certificates for this new Microsoft Azure virtual machine.

8. Specify the target database name if different from the source database name. If the target database already exists, the system will automatically increment the database name rather than overwrite the existing database.

9. Click Next and then click Finish to start the migration.

The wizard takes a compressed backup of the database to the specified backup location, and the Cloud Adaptor service within the VM is used to restore the backup into the target SQL instance in Azure VM.

The time taken by the wizard is dependent on the backup size and time to copy the compressed backup across network and restore time, which in turn is dependent on the Azure VM Compute Size.

If you created a new virtual machine, configure the Azure virtual machine and the SQL Server instance by following the steps mentioned in the previous chapter.

Perform a Compressed Backup On-Premise and Copy the Files to Azure VM

Use this method when you cannot use the Deploy a SQL Server Database to a Microsoft Azure VM wizard either because you are migrating to a version of SQL Server prior to SQL Server 2014 or your backup file is larger than 1 TB. If your backup file is larger than 1 TB, you must stripe it because the maximum size of a VM disk is 1 TB. Use the following general steps to migrate a user database using this manual method:

1. Perform a compressed full database backup to an on-premise location.

2. Copy your backup file(s) to your VM using remote desktop, Windows Explorer, or the copy command from a command prompt.

3. Restore the database from the copied backup file to the SQL Server instance in Azure VM.

Perform a Backup to URL and Restore It from URL

This method is no different from the previous method except for the fact that data is backed up directly to Azure Blob storage and is restored from the same location. The following article provides the steps to create a backup to Azure Blob storage URL:

https://msdn.microsoft.com/library/dn435916.aspx

Microsoft introduced the backup to URL option starting with SQL 2012 SP1 CU2; this method can only be used when the source SQL Server instance is already on or above SQL 2012 SP1 CU2.

Use the backup to URL method when you cannot use the Deploy a SQL Server Database to a Microsoft Azure VM wizard because your backup file is larger than 1 TB and you are migrating from and to SQL Server 2016. For databases smaller than 1 TB or running a version of SQL Server prior to SQL Server 2016, use of the wizard is recommended. With SQL Server 2016, striped backup sets are supported, recommended for performance, and required to exceed the per-blob size limits. For very large databases, the use of the Windows Import/Export Service is recommended.

Detach the Database, Copy to URL, and Attach It from URL

Use this method when you plan to store these files using the Azure Blob storage service and attach them to SQL Server running in an Azure VM, particularly with very large databases. Use the following general steps to migrate a user database using this manual method:

1. Detach the database files from the on-premise database instance.
2. Copy the detached database files into Azure Blob storage using the AZCopy command-line utility.
3. Attach the database files from the Azure URL to the SQL Server instance in the Azure VM.

Convert to VM, Upload to URL, and Deploy as New VM

This method is the same as the one we discussed in the section Creating a Sysprepped Image of SQL Server in Hyper-V and Adding It to Azure. This method is useful when you want to bring your own license and migrate the entire SQL instance, along with the user and system databases, to Azure VM.

Ship Hard Drive Using Windows Export/Import Service

The Microsoft Azure team introduced the Windows Import/Export service method a few years ago as an efficient solution for transferring large volumes of on-premise data into Azure Blob storage. This method is used when the volume of data to be transferred is too high and transferring it over the network may be prohibitive or may otherwise be unfeasible. With the Windows Import/Export service, you can ship TBs of encrypted data via hard drives through FedEx to Azure data centers. The requirements for this method are only a 3.5 inch SATA II/III hard drive with a maximum size of up to 6 TB. Additionally, the hard drive needs to be BitLocker encrypted before sending it to Azure data centers. The detailed steps to transfer the data using the Windows Import/Export Service is described in the following article:

https://azure.microsoft.com/en-us/documentation/articles/storage-import-export-service/

Use Always On to Add a Replica in Azure and Failover to Migrate

This method is useful when you want to minimize the failover time for migration. In this method, we add the SQL Server instance on Azure VM as an Azure replica of the on-premise SQL Server instance using the new Add Azure Replica Wizard. This method requires a hybrid-IT environment where an on-premise subnet has a site-to-site VPN with Windows Azure. After the always-on availability group is set up and synchronized, you can failover at any time and go live with the database on Azure. The on-premise database can then be removed from the availability group and decommissioned once the migration is successful.

Chapter 4 Performance Considerations for SQL Server on Azure VMs

The performance considerations for running SQL Server on Azure VMs are no different from SQL Server running on a box, but there are various other optimizations or considerations available in Azure infrastructure, which you need to be aware of and leverage to give you predictable performance at reduced cost. In this chapter, we will discuss these best practices for performance as well as considerations for running SQL Server on Azure VMs.

Storage Considerations

One of the major differences when running SQL Server on Azure VMs is that you no longer have storage admins to work with. This can be both good and bad, since you can now take control of the storage yourself and have no dependencies on the storage team, but now, if the storage doesn't give you the best performance, you do not have anyone else to blame. In this section, we will discuss the various storage considerations and best practices you need to be aware of to give you an optimized performance in Azure.

Be Aware of Storage Account Limits

As mentioned earlier, Azure is a shared, multi-tenant environment, and hence all the services on Azure have limitations to ensure each account gets dedicated, predictable storage and compute performance without impacting other accounts. The storage account acts a container that holds the operating system disk, temporary disks, and data disks, as well as Azure Blob storage. There are limitations on the maximum number of IOPs and ingress/egress limits per storage account. Ingress refers to all the data sent into the storage account, while egress refers to all the data extracted from the storage account.

The following table provides the list of Standard and Premium storage account limitations:

Standard Storage Limits

Table 2: Standard Storage Account Limits

Resource	Default Limit
Max number of storage accounts per subscription	100
TB per storage account	500 TB

Resource	Default Limit
Max size of a single blob container, table, or queue	500 TB
Max number of blocks in a block blob	50,000
Max size of a block in a block blob	4 MB
Max size of a block blob	50,000 X 4 MB (approx. 195 GB)
Max size of a page blob	1 TB
Max size of a table entity	1 MB
Max number of properties in a table entity	252
Max size of a message in a queue	64 KB
Max size of a file share	5 TB
Max size of a file in a file share	1 TB
Max number of files in a file share	Only limit is the 5 TB total capacity of the file share
Max number of blob containers, blobs, file shares, tables, queues, entities, or messages per storage account	Only limit is the 500 TB storage account capacity
Max 8 KB IOPS per persistent disk (Basic Tier virtual machine)	300

Resource	Default Limit
Max 8 KB IOPS per persistent disk (Standard Tier virtual machine)	500
Total Request Rate (assuming 1KB object size) per storage account	Up to 20,000 IOPS, entities per second, or messages per second
Target Throughput for Single Blob	Up to 60 MB per second, or up to 500 requests per second
Target Throughput for Single Queue (1 KB messages)	Up to 2000 messages per second
Target Throughput for Single Table Partition (1 KB entities)	Up to 2000 entities per second
Target Throughput for Single File Share (Preview)	Up to 60 MB per second
Max ingress[3] per storage account (US Regions)	10 Gbps if GRS/ZRS enabled, 20 Gbps for LRS
Max egress[3] per storage account (US Regions)	20 Gbps if GRS/ZRS enabled, 30 Gbps for LRS
Max ingress per storage account (European and Asian Regions)	5 Gbps if GRS/ZRS enabled, 10 Gbps for LRS
Max egress per storage account (European and Asian Regions)	10 Gbps if GRS/ZRS enabled, 15 Gbps for LRS

Premium Storage Account Limits

Resource	Default Limit
Total disk capacity per account	35 TB
Total snapshot capacity per account	10 TB
Max bandwidth per account (ingress + egress1)	>50 Gbps

Note: The above limitations are as of today and might change in the future. The above table and up-to-date limitations on Storage Account and other Azure services can be found in the following article: https://azure.microsoft.com/en-us/documentation/articles/azure-subscription-service-limits/#storagelimits

As seen in the previous table, the total request rate supported by a storage account is 20,000 IOPs. If we have a VM which is running the disk at full capacity, the maximum number of disks from a Standard storage account which can be attached to the VM is 40 (20000/500 IOPs) to give you the desired performance. However this is not a hard limitation, so if you exceed this value the storage account throttling can kick in and can have a negative performance impact on your VMs.

It is recommended multiple, separate storage accounts be used depending on the workload to avoid throttling and negative impact on the performance of SQL Server due to storage latencies. It is important that the storage account is in the same location as the VMs to obtain the best performance.

Disable Geo-Redundancy on Storage Accounts

Azure Storage accounts support geo-redundancy to replicate the storage contents across multiple data centers across regions. This provides High Availability and Disaster Recovery (HA/DR) and high performance for Content Delivery Networks (CDN) and similar applications. However, geo-redundancy in a storage account doesn't guarantee write-consistency across disks, which makes it practically unusable for SQL Server as an HA/DR solution in Azure since it breaks the Write Ahead Logging (WAL) protocol for SQL Server and results in corruption in the database on the DR site.

Since geo-redundancy is unusable for SQL Server workloads, it is beneficial to disable geo-redundancy for storage accounts hosting the Data disks for SQL Server to avoid replication overhead and reduce the per-minute costs for storage accounts. The recommended options for creating a storage account for SQL Server is locally redundant storage (LRS), which replicates the storage content locally in Azure data centers via three redundant copies for better availability and reliability. The storage account can be created from Azure portal or PowerShell.

Use Premium Storage

As of today, Microsoft Azure provides two types of storage offerings:

- Standard Storage
- Premium Storage

Azure Standard Storage supports up to 500 IOPs per data disk attached to the VMs. The IOPs supported by Azure Standard Storage Data Disk can be further bumped up by adding multiple disks to the storage pool, which acts a software RAID configuration. However, the Standard Storage disks don't provide the low latency desired for running SQL Server workloads. The high latency of the disk makes it an unattractive offering for running SQL Server workloads, since the SQL Server has an inherently high I/O intensive workload with high read/write for reading and writing the data into the database and log files.

Azure Premium Storage supports up to 50k IOPs per data disk attached to the VMs. Moreover, it supports high IOPs at reduced disk latency, which makes it a preferred choice for high-performance, I/O-intensive workloads running on Azure Virtual Machines and more specifically for SQL Server running on Azure VMs.

In order to use Premium Storage, you need to create a Premium Storage account. This can be created from the Azure portal or through PowerShell. Azure uses the storage account as a container for your operating system (OS) and data disks. If you want to use a Premium Storage account for your VM disks, you need to use the DS-series of VMs. You can use both Standard and Premium storage disks with the DS-series of VMs. But you cannot use Premium Storage disks with non-DS-series VMs. The amount of IOPs and bandwidth supported by Premium Storage data disks depends on the Premium Storage disk types. As of today, Premium Storage supports three types of disk types, namely P10, P20, and P30. The IOP and bandwidth supported by each of these types is listed in the table below. This table can also be found at the following Microsoft article: https://azure.microsoft.com/en-us/documentation/articles/storage-premium-storage-preview-portal/#using-premium-storage-for-disks/.

Table 3: Premium Storage Account Disk Types

Premium Storage Disk Type	P10	P20	P30
Disk size	128 GB	512 GB	1024 GB (1 TB)
IOPS per disk	500	2300	5000
Throughput per disk	100 MB per second	150 MB per second	200 MB per second

Note: The throughput and IOPs supported by a disk in a VM are also dependent on the size of the VM (DS0-DS14). It is important to ensure and choose a VM size which supports sufficient bandwidth to run these disks at their maximum capacity.

Further details on Azure Premium Storage can be found in Microsoft's article on the subject:

https://azure.microsoft.com/en-us/documentation/articles/storage-premium-storage-preview-portal/#using-premium-storage-for-disks

Use Separate Disks for Data, Log Files, and Tempdb

For SQL Server workloads, it is recommended to separate disks for data files and log files due to the nature of the IO (random read/write IO versus sequentially written IO, respectively). When using Premium Storage, the recommendation for SQL Server workloads running on Azure VMs is at minimum 2 X P30 disks to be attached to the VMs dedicated for user data files and log files. When using Standard Storage with storage pools—as discussed in the following section—it is recommended to create at least two storage pools to separate the IO workload for data and log files. Further operating system disks and temporary disks shouldn't be used for storing any database files (including system databases except for tempdb, which is discussed in the following section), backups, error logs, or tracing files. In other words, the OS disk should not be used to host any SQL Server files except for the SQL Server binaries.

Use Temporary Disk to Store Tempdb or BPE in D or G-series VMs

D and G-series VMs have a temporary drive, labeled as *D:,* along with an OS drive attached to the VM when the VM is provisioned. It is called a temporary disk because the data stored in the disk is not persistent and is lost on every reboot of the VM. The temporary disk is a high performance, low latency SSD drive, but it cannot be used to store user data or log files since the data is not persistent. However, the temporary disk can be used for storing tempdb or Buffer Pool extension (BPE) files which get cleaned up or recreated on every restart of SQL Server service.

Tempdb is one of the most highly accessed database files in a SQL Server instance since most queries require tempdb for sorting, hashing, versioning, or storing temp tables. Moving tempdb to SSD disks would improve the overall performance of the SQL Server workload, especially the ones that use tempdb objects heavily.

The detailed steps to move tempdb and BPE files to temporary disks in a D-series VM is documented in the following blog by the Microsoft Product Team:

http://blogs.technet.com/b/dataplatforminsider/archive/2014/09/25/using-ssds-in-azure-vms-to-store-sql-server-tempdb-and-buffer-pool-extensions.aspx

Using Storage Spaces for Data Disks in Standard Storage

As discussed in the previous section, it is recommended to use Premium Storage accounts for configuring SQL Server on Azure VMs. If for some reason you would prefer to use Standard Storage for configuring data disks for SQL Server VMs, it is recommended you use Storage Spaces to stripe the data across multiple data disks. The advantage of creating storage pools and using Storage Spaces is that it allows you to exceed the per-disk IOP. In Standard Storage, the max IOPs supported per-disk is 500 IOPs, but you can stripe four data disks together in a storage pool to support close to 2000 IOPs per LUN. The storage pool improves the overall IO throughput as well as the latency of the storage to some extent.

It is, however, important to note that Storage Spaces was introduced with Windows8/Windows Server 2012. Only VMs running Windows 8/Windows Server 2012 can support the creation of storage pools and storage spaces to stripe across multiple data disks.

For SQL Server workloads, it is recommended to create, at minimum, two storage pools per VM, one for data files and the other to host the log files. Set stripe size to 64 KB for OLTP workloads and 256 KB for data warehousing workloads to avoid performance impact due to partition misalignment. In addition, set column count equal to the number of physical disks. To configure a storage space with more than eight disks, you must use PowerShell (not Server Manager UI) to explicitly set the number of columns to match the number of disks. For more information on how to configure Storage Spaces, see Storage Spaces Cmdlets in Windows PowerShell.

The following PowerShell code snippet can be used for creating storage pools in SQL Server Azure VMs. The code shows the creation of two storage pools, each striped across two disks, each with a stripe size of 64KB. The storage spaces are labelled as *DataDisks* and *LogDisks* to identify them as drives for data and log files, respectively.

Code Listing 16: Creating Storage Pool for SQL Server Data and Log Files in Azure VM

```
# The original script for Provisioning of SQL Server Azure VM was created
by Sourabh Agrawal and Amit Banerjee

# The script is modified by Parikshit Savjani to suit the readers of the
book SQL on Azure

$PoolCount = Get-PhysicalDisk -CanPool $True

$PhysicalDisks = Get-PhysicalDisk | Where-Object {$_.FriendlyName -like
"*2" -or $_.FriendlyName -like "*3"}


New-StoragePool -FriendlyName "DataFiles" -StorageSubsystemFriendlyName
"Storage Spaces*" -PhysicalDisks $PhysicalDisks |New-VirtualDisk -
FriendlyName "DataFiles" -Interleave 65536 -NumberOfColumns 2 -
ResiliencySettingName simple –UseMaximumSize |Initialize-Disk -
PartitionStyle GPT -PassThru |New-Partition -AssignDriveLetter -
UseMaximumSize |Format-Volume -FileSystem NTFS -NewFileSystemLabel
"DataDisks" -AllocationUnitSize 65536 -Confirm:$false
```

```
PhysicalDisks = Get-PhysicalDisk | Where-Object {$_.FriendlyName -like "*4"
-or $_.FriendlyName -like "*5"}


New-StoragePool -FriendlyName "LogFiles" -StorageSubsystemFriendlyName
"Storage Spaces*" -PhysicalDisks $PhysicalDisks |New-VirtualDisk -
FriendlyName "LogFiles" -Interleave 65536 -NumberOfColumns 2 -
ResiliencySettingName simple –UseMaximumSize |Initialize-Disk -
PartitionStyle GPT -PassThru |New-Partition -AssignDriveLetter -
UseMaximumSize |Format-Volume -FileSystem NTFS -NewFileSystemLabel
"LogDisks" -AllocationUnitSize 65536 -Confirm:$false
```

Caching Policy for Disks

If you are using Premium Storage for your data and log files, it is recommended you disable caching for disks hosting the log files and enable read caching on the disks hosting data files and tempdb. With Standard Storage, it is recommended you disable caching for all data disks.

The default caching policy for the OS disk is read/write, which doesn't need to be changed. It is highly recommended that you do not store any database files, including the system databases, error logs, backup, and tracing files, on the OS disk.

NTFS Allocation Unit Size

All the disks attached to the VMs running SQL Server should be formatted with NTFS with an NTFS allocation unit size of 64.

Choose the Right VM Size for Your Workload

As discussed earlier, the VM size decides the compute and storage capacity supported by the VM. In Azure, VMs are available in two tiers: basic and standard. Both types offer a choice of sizes, but the basic tier doesn't provide some capabilities available with the standard tier, such as load-balancing and auto-scaling. The standard tier supports VM from A-series, D-series, DS-series, and G-series, which were discussed earlier. The detailed sizes of VM available in Azure is documented in the following article from Microsoft:

https://azure.microsoft.com/en-us/documentation/articles/virtual-machines-size-specs/

When working with SQL Server on Azure, it is recommended you use Premium Storage for better performance. As discussed earlier, if you need to use Premium Storage for your VMs, you need to use DS-series VMs. DS-series VMs support attaching disks from both Premium Storage accounts as well as Standard Storage accounts, which makes them the preferred choice for installing and configuring SQL Server. Even if you are planning to attach Standard Storage to the VMs, it might make more sense to use DS-series VMs, since DS-series VMs would allow you to migrate to Premium Storage with minimal efforts if you chose to upgrade later.

For SQL Server workloads it is therefore recommended you choose the Standard tier with DS-series VM. The generic recommendation is running a VM size of DS3 or higher for SQL Enterprise edition and DS2 or higher for SQL Standard Edition.

The size of DS-series VMs (DS1 – DS14) you choose depends on the application workloads, throughput requirements, and resource utilization on the VM. The best part of running the VM on the cloud is you can always start with the smaller VM size and scale up to a higher size with minimal effort, cost, and downtime.

Use SQL Server Best Practices

Most performance recommendations or considerations for running a SQL Server in Azure VMs are focused on storage and IO optimization since storage is the major performance differentiator between running a SQL Server on-premise versus running a SQL Server in Azure VMs. In this section, we will discuss some of the known SQL Server best practice configurations which optimize the IO, thereby helping optimize the overall performance of SQL workload running in Azure VMs.

Use Lock Pages in Memory

Lock Pages in Memory is a system security privilege that is not assigned to any users, including administrators, by default. This is because any application or service that runs with an account with this privilege can lock their page in memory, and even OS cannot page out these buffers from memory. For SQL Server workloads, the SQL buffer pool pages should be locked in memory and shouldn't be paged out by OS to give us predictable performance. It is recommended you assign Lock Page in Memory privilege to the SQL service startup account. However, when using the Lock Pages in Memory privilege, it is critical to cap the max server memory of SQL Server instances to around 70-80% of total memory on the VM to leave breathing space for OS and other applications or services (monitoring agents, backup software, SQL Management Studio) on the VM.

You can use the following steps to enable Lock Pages in Memory privilege for the SQL Server service account:

1. On the **Start** menu, click **Run**. In the **Open** box, type **gpedit.msc**.

2. On the **Local Group Policy Editor** console, expand **Computer Configuration**, and then expand **Windows Settings**.

3. Expand **Security Settings**, and then expand **Local Policies**.
4. Select the **User Rights Assignment** folder.
5. The policies will be displayed in the details pane.
6. In the pane, double-click **Lock pages in memory**.
7. In the **Local Security Setting – Lock pages in memory** dialog box, click **Add User or Group**.
8. In the **Select Users, Service Accounts, or Groups** dialog box, add an account with privileges to run sqlservr.exe.
9. Log out and then log back in for this change to take effect.

Note: On the VM server, if you assign the Lock Pages in Memory privilege to the SQL service account and you are running any other service using the same account, the memory pages committed by the other service are also locked and cannot be paged out. From a security and administrative standpoint, it is recommended to keep the SQL service account dedicated to and exclusively for running the SQL Server service.

Enable Instant File Initialization for Data Files

Instant file initialization was first introduced by the Microsoft SQL Product Team in SQL 2005 to reduce the time required for initial file allocation or growth for data files when the database is created or grown. Instant file initialization skips the zeroing of the files, which is an IO intensive operation, and reduces the overall time for new file allocation significantly. To take advantage of instant file initialization, you need to grant permission to the SQL Server (MSSQLSERVER) service account with SE_MANAGE_VOLUME_NAME and add it to the Perform Volume Maintenance Tasks security policy. When we provision a SQL Server image from Azure Gallery, this privilege is not automatically assigned to the SQL service account and needs to be manually assigned as a post-configuration task after provisioning the SQL Server image.

You need to take the following steps to grant an account the perform volume maintenance tasks permission:

1. On the computer where the backup file will be created, open the **Local Security Policy** application (secpol.msc).
2. In the left pane, expand **Local Policies** and then click **User Rights Assignment**.
3. In the right pane, double-click **Perform volume maintenance tasks**.
4. Click **Add User or Group** and add any user accounts that are used for backups.
5. Click **Apply**, and then close all Local Security Policy dialog boxes.
6. Restart SQL Server services for the permissions to take effect.

Note: Using instant file Initialization can have security implications. You can read the following article from MSDN to learn more about instant file initialization: https://msdn.microsoft.com/library/ms175935.aspx

Instant File Initialization is supported and available only for data files. Even when you assign this privilege to your SQL Service account, the T-log file allocation will be IO intensive and time consuming since it will involve the zeroing of files.

Disable Autoshrink and Avoid Autogrow

Autoshrink is a database file property that allows you to shrink a database automatically when the database is not in use. Autoshrink is an IO intensive activity that causes fragmentation of database files as well as the disks. The Autoshrink property is disabled by default, and it is recommended it remain disabled; it shouldn't be used as a preferred method for shrinking the database files, especially when running on Azure VMs.

Autogrow is a database file property used to grow the database automatically when the database files are running out of space. Autogrow is also an IO-intensive operation, and can kick off randomly at any time when the data or log file is running out of space. Autogrow causes fragmentation of the database files and disk, and if it occurs during business hours it can stall or hang the SQL Server until the database file growth is completed. It is not recommended to disable autogrow, since it can work as a form of insurance and help you in critical situations to grow the database file when it is full. Ideally, you should have a monitoring and alert system which alerts a DBA when a database file has only 10-15% of free space on the file. The DBA should respond to the alert by growing the file manually during maintenance windows or turning it off during business hours to avoid autogrow.

Use Database Page Compression

SQL Server data compression was introduced with SQL 2008 and allows you to compress the database pages while storing them in the data files. This reduces the IO bandwidth and storage required to store the data files and thereby improves the overall performance due to reduced IO workload. SQL Server supports row compression, page compression, and columnstore compression (introduced in SQL 2014), each of which provides a different compression ratio at the expense of higher CPU utilization.

For further details on data compression in SQL Server, you can refer to following article from MSDN:

https://msdn.microsoft.com/library/cc280449.aspx

For SQL Server workloads running on Azure VM, it is recommended you use database compression wherever possible to reduce IO bandwidth and thereby improve the response time of the queries.

Backup to Azure Blob Storage with Compressed Backups

SQL Server supports native backup compression starting with SQL 2008 R2, which helps to reduce the size of the backups as the backups are being created. For a SQL Server workload running on a box or Azure VM, it is recommended to turn on backup compression at the instance level. This will ensure any backups performed on the SQL instance are compressed. Furthermore, the restoration of the database is not impacted much when the backups are compressed.

You can use the following t-sql code to enable compressed backups at the instance level:

Code Listing 17: Enable Compressed Backups

```
USE master;

GO

EXEC sp_configure 'backup compression default', '1';

RECONFIGURE WITH OVERRIDE;
```

When performing backups for SQL Server running in Azure virtual machines, you can use SQL Server Backup to URL. This feature is available starting with SQL Server 2012 SP1 CU2 and is recommended for backing up to the attached data disks. When you back up or restore to/from Azure storage, follow the recommendations provided at SQL Server Backup to URL Best Practices and Troubleshooting and Restoring from Backups Stored in Azure Storage. You can also automate these backups using Automated Backup for SQL Server in Azure Virtual Machines.

Apply SQL Server performance fixes

By default, it is always recommended to run SQL Server on the latest service pack or cumulative update to cover all known issues for the server. Specifically for SQL Server 2012 running on Azure VMs, install Service Pack 1 Cumulative Update 10. This update contains the fix for poor performance on I/O when you execute the select into temporary table statement in SQL Server 2012.

Chapter 5 Business Continuity Solutions for SQL Server on Azure VMs

High-Availability Disaster Recovery (HADR) Solutions for SQL Server in Azure

One of the prime concerns of most operations teams is business continuity and high availability of the application and the database in the event of hardware failure, infrastructure failure, or site failure. SQL Server on Azure VMs supports all the high-availability and disaster recovery solutions that are available for on-premise versions of SQL Server except failover clustering, which is not supported in Azure yet.

As discussed earlier, while Azure storage supports geo-redundancy and replication of storage across Azure data centers, it is not consistent across disks, which makes it unusable as an HADR solution for SQL Server in Azure. Further while Azure storage maintains locally redundant copies of the database, it is not a substitute for backups or HADR solutions for SQL Server. Microsoft strongly recommends setting up an HADR solution for SQL Server databases running on Azure depending on Recovery Point Objective (RPO) and Recovery Time Objective (RTO) of the databases.

SQL Server HADR technologies that are supported in Azure include:

- AlwaysOn Availability Groups
- Database Mirroring
- Log Shipping
- Backup and Restore with Azure Blob Storage Service

Further using the above HADR technologies, it is also possible to set up hybrid IT solutions where an on-premise version of SQL Server replicates data to a SQL Server instance running on Azure VMs. The hybrid solution gives you the best of both worlds, where the SQL Server instance running on the dedicated, high-performance on-premise box replicates the data to a low-cost VM running on the Azure data center. The secondary instance running on Azure can be scaled up to a higher configuration to support production workload in the event of failure of a primary node. Hence the hybrid solution can help you provide a cost-efficient, highly available, and highly performant database solution using the SQL Server HADR solutions. A hybrid deployment requires a site-to-site VPN setup between an on-premise data center and Azure data center with Azure Network to allow connectivity and data replication over a secure VPN tunnel.

In this chapter, we will discuss the setup and configuration of AlwaysON Availability Groups and Backup/Restore in Azure VMs. Database Mirroring is deprecated and Microsoft strongly recommends the use of AlwaysON Availability Group as an HADR solution, which is the successor of Database Mirroring and provides more features than that provided by mirroring. The setup and configuration of Database Mirroring and Log shipping is similar to their corresponding on-premise setups and hence will be not be discussed in this chapter.

AlwaysON Availability Groups in Azure

The AlwaysOn Availability Groups are a high-availability and disaster-recovery solution that provides an enterprise-level alternative to database mirroring. Introduced in SQL Server 2012, AlwaysOn Availability Groups maximize the availability of a set of user databases for an enterprise. An availability group supports a failover environment for a discrete set of user databases, known as availability databases, that failover together as a group. An availability group supports a set of read-write primary databases and one to four sets of corresponding secondary databases in SQL 2012 and up to eight sets starting with SQL 2014. Optionally, secondary databases can be made available for read-only access and/or some backup/checkdb operations. An availability group fails over at the level of an availability replica. Failovers are not caused by database issues such as a database becoming suspect due to a loss of a data file, deletion of a database, or corruption of a transaction log.

In AlwaysON Availability Group, each participating SQL instance referred as an availability replica is installed separately as a standalone instance on a separate node. The participating nodes or servers are part of the same Windows Server Failover Cluster (WSFC) but have their own dedicated storage attached, allowing the two nodes to be geographically dispersed from each other. WSFC is required for arbitration and to support automatic failover, as discussed later in this section.

AlwaysON Availability Group supports the following modes of operation:

Asynchronous-Commit Mode

In this mode, when a transaction is committed on the primary replica (active SQL instance), the transaction is flushed immediately to the transaction log of primary databases on the disk and simultaneously placed into the network queue to replicate the logs to the secondary replica. The commit doesn't wait for the acknowledgement from the secondary replica and proceeds with the next set of transactions. This availability mode is a disaster-recovery solution that works well when the availability replicas are distributed over considerable distances since the performance of the transactions are not dependent on the network latency of the data replication.

Synchronous-Commit Mode

In this mode, as with the asynchronous-commit mode, when a transaction is committed on the primary replica (active SQL instance), the transaction is flushed immediately to the transaction log of primary databases on the disk and simultaneously placed into the network queue to replicate the logs to the secondary replica. However, the commit transaction waits for the acknowledgement from the secondary replica and proceeds with the next set of transactions only after the ACK is received from the remote replica. This availability mode emphasizes high availability and data protection over performance, at the cost of increased transaction latency. A given availability group can support up to one sync-commit replica in SQL 2012 and up to three synchronous-commit availability replicas in SQL 2014, including the current primary replica.

Synchronous-Commit Mode with Automatic Failover

This mode is a variation of synchronous-commit mode wherein the availability group is configured for automatic failover. With automatic failover, the cluster service performs IsAlive and LooksAlive checks on the SQL instance to detect any failure. If the failure condition is met, the availability group resource in the cluster is failed over to the secondary replica. Automatic failover is supported only with synchronous-commit mode because the two replicas are transaction-consistent, while asynchronous-commit mode only supports manual failover, also known as forced failover, which might lead to some data loss.

AlwaysON Availability Group Topologies in Azure

AlwaysON Availability Group in Azure can be configured for the following four topologies depending on the location of the availability replicas and intent for using it as a HA solution, DR solution, HADR solution, or setting up a hybrid solution.

High Availability Solution for SQL Server in Azure

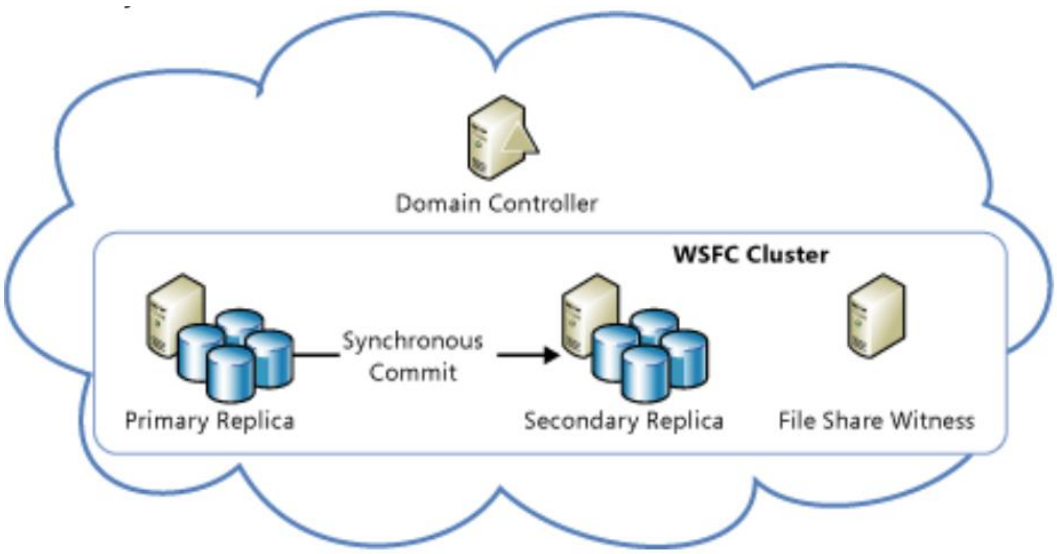

Figure 6: AlwaysON Availability Groups as HA Solution in Azure

When configuring availability groups as a high-availability solution for SQL Server in Azure, both participating SQL Server VMs reside in the same Azure data centers as a part of the same availability sets in Azure. Since the two VMs reside in the same data centers, AlwaysON Availability Group can be configured in synchronous-commit mode with automatic failover since network latency is minimal. Availability sets in Azure enable you to place the high-availability VMs into separate Fault Domains (FDs) and Update Domains (UDs). This will ensure failure of one of the VMs in the data center due to hardware or infrastructure failure will not impact other VMs in the same availability set and vice-versa, since it will be running on separate, fault-tolerant hardware in the same data center. For Azure VMs to be placed in the same availability set, you must deploy them in the same cloud service. Only nodes in the same cloud service can participate in the same availability set.

Azure VMs are added to a cloud service and availability set at the time of provisioning the Azure VM discussed in Chapter 2. In this case, the cloud service and availability set is created first, and at the time of provisioning the VM, it is mapped and associated with the cloud service and availability set.

The following PowerShell code snippet shows the creation of a cloud service and new availability set and the provisioning of the two VMs participating in AlwaysON Availability Group as a part of the same cloud service and availability set.

Code Listing 18: Provisioning Two VMs for AlwaysON Availability Group as Part of the Same Cloud Service and Availability Set

```
New-AzureService -ServiceName $ServiceName -Location $Location|Out-Null


New-AzureAvailabilityset  -Name $AvSet -Location $Location|Out-Null


New-AzureQuickVM -Windows -ServiceName $ServiceName -AvailabilitySetName
$AvSet -Name $VM1 -ImageName $ImageName.imagename -Password $password -
AdminUsername $AdminAccount -InstanceSize $VmSize -EnableWinRMHttp | out-
null

New-AzureQuickVM -Windows -ServiceName $ServiceName -AvailabilitySetName
$AvSet -Name $VM2 -ImageName $ImageName.imagename -Password $password -
AdminUsername $AdminAccount -InstanceSize $VmSize -EnableWinRMHttp | out-
null
```

While provisioning the VMs for configuring AlwaysON Availability Group as a high-availability solution, all the VMs, the cloud service, the availability set, and storage for the VMs are created in the same location which resides in the same Azure data center.

If the VM was not added to the availability set or was added to a different availability set at the time of provisioning, and if it is later decided that the VM needs to participate in AlwaysON Availability Group, the VM can be added to the availability set after provisioning using Azure Management Portal or using the Set-AzureAvailabilitySet cmdlet, as shown below

Code Listing 19: Adding a VM to Availability Set after Provisioning

```
Get-AzureVM -ServiceName $ServiceName -Name "vm1" | Set-
AzureAvailabilitySet -AvailabilitySetName $Avset |Update-AzureVM
```

After the VM is updated, it needs to be shut down (deallocated) and restarted to start the VM as a part of specified availability set.

You can also add a VM to the availability set using Azure Management Portal. The following article describes the steps to achieve the same using Azure Management portal:

https://azure.microsoft.com/en-us/documentation/articles/virtual-machines-how-to-configure-availability/

Disaster Recovery Solution for SQL Server in Azure

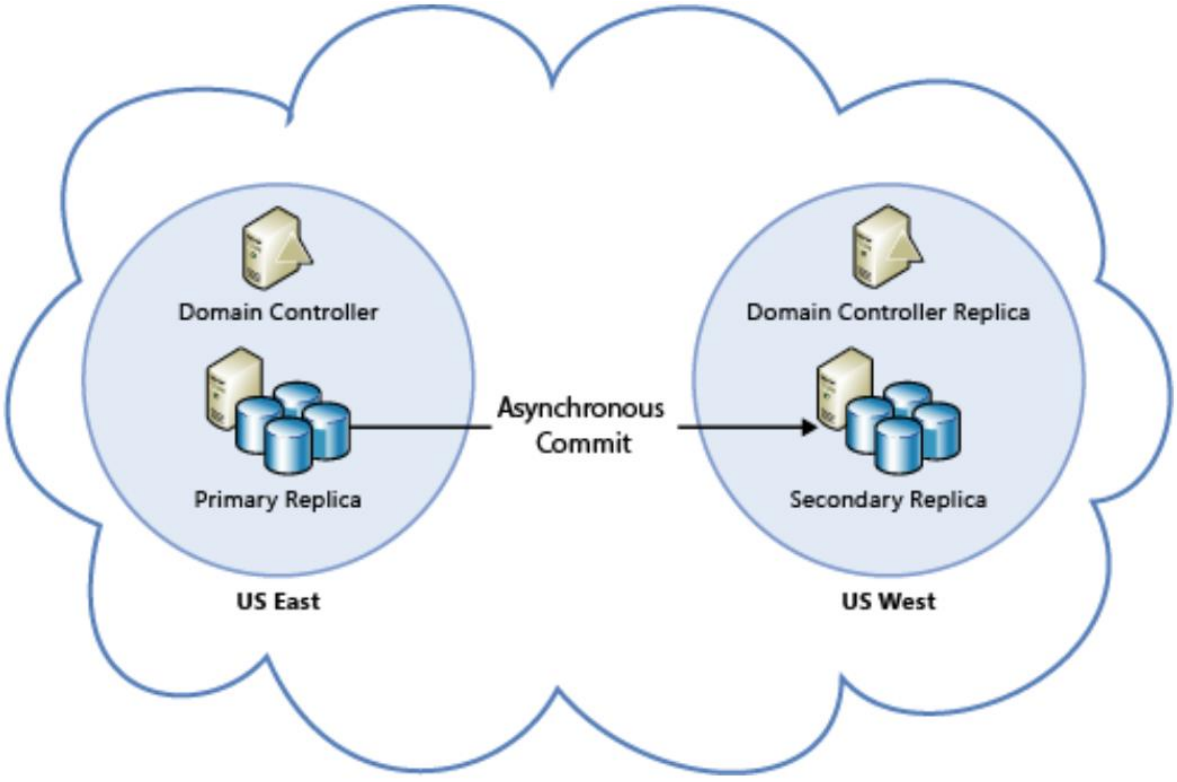

Figure 7: AlwaysON Availability Group as a DR Solution in Azure

When configuring availability groups as a disaster recovery solution for SQL Server in Azure, each participating SQL Server VM resides in a different Azure data center, which might be in the same country (e.g., South Central US and East US) or in different countries (e.g. East US and Europe). Since the two participating VMs in AlwaysON Availability Groups are in different data centers, sync-commit mode is not recommended due to high network latency. Asynchronous commit mode is the preferred mode for this configuration. Within a data center or Azure location, all replicas should be within the same cloud service, same availability set, and same VNet as discussed earlier. A cloud service and VNet cannot stretch across Azure data centers or locations, so the participating VMs in different data centers are mapped to separate cloud services and separate VNets. This configuration requires Azure VNet-to-VNet connectivity, which can be configured by setting up a site-to-site VPN tunnel or ExpressRoute. Availability set configuration doesn't apply to these VMs since the VMs already reside in separate data centers and thus do not necessarily have to be part of any availability set.

High Availability and Disaster Recovery for SQL Server in Azure

This configuration is a combination of the previous two configurations. AlwaysON Availability groups allow up to four availability replicas in SQL 2012 and 8 replicas starting with SQL 2014. It is possible to add multiple availability replicas in sync-commit mode to the same Azure data center to provide high availability and configure one or more replicas in asynchronous-commit mode in a separate data center to provide disaster recovery. Furthermore, the secondary replicas can be opened in read-only mode, which can used to offload reporting and backups. In this configuration, all participating Azure VMs residing in the same location are part of the same cloud service, availability set, and VNet, while the remote replicas are part of the separate cloud service, availability set and VNet. As discussed earlier, this configuration would require VNet-to-VNet connectivity.

Hybrid IT: Disaster Recovery Solution for On-Premise SQL Server in Azure

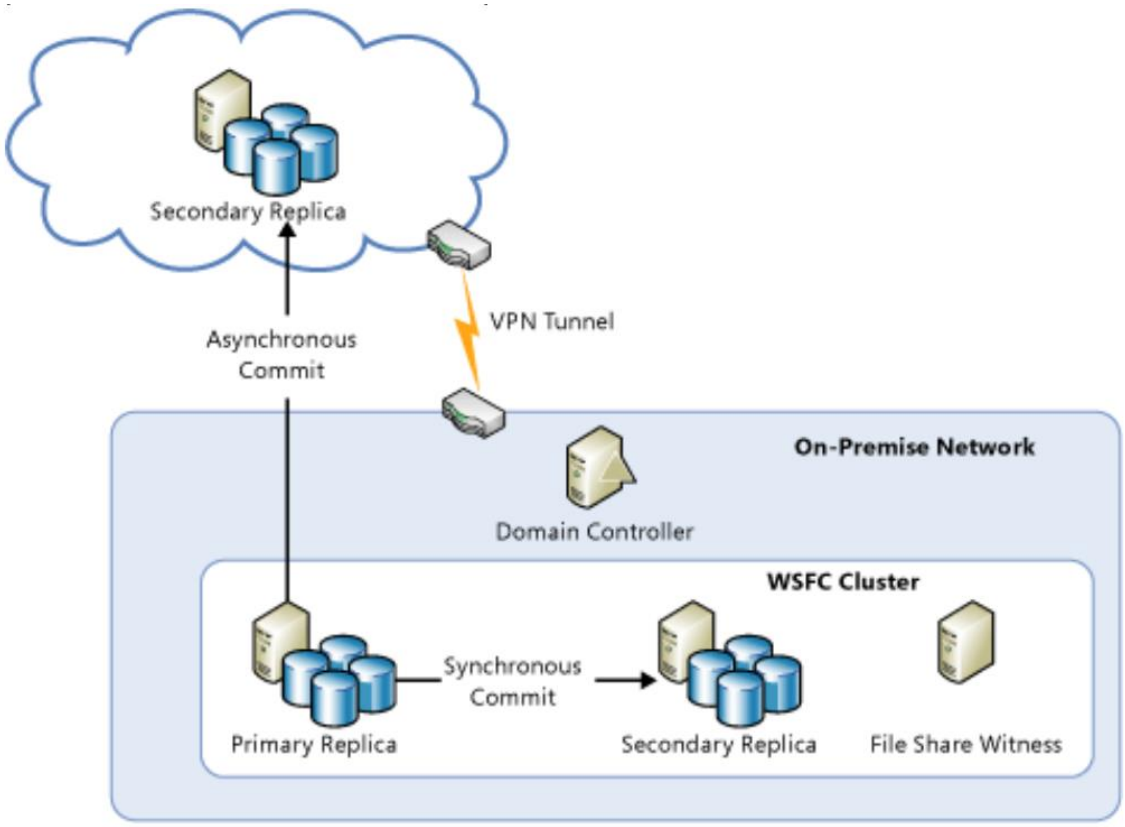

Figure 8: Hybrid IT: DR Solution for On-Premise SQL Server in Azure

This configuration allows you to set up the most cost-efficient disaster recovery solution for an on-premise instance of SQL Server by configuring the secondary replica of the availability group in an Azure VM. In this configuration, all on-premise replicas can be configured in synchronous-commit mode while the secondary replica in Azure is configured in asynchronous-commit mode. This configuration again requires VNet-to-VNet connectivity between on-premise data centers and Azure data centers. This connectivity can be configured by setting up a site-to-site VPN tunnel by using ExpressRoute or following the procedure as described in the following article:

https://azure.microsoft.com/en-us/documentation/articles/vpn-gateway-site-to-site-create/

Preparing Azure Environment for AlwaysON Availability Group

In order to set up and configure AlwaysON Availability Group in Azure VMs in order to implement one of the previously discussed topologies, it is important to first meet the following pre-requisites:

1. Create an Azure Virtual Network (Azure VNet). While creating an Azure Virtual Network, you can set up DNS servers and VPN connectivity to another site. This will be required if you are planning to configure AlwaysON Availability Group to implement DR, HADR, or Hybrid IT solutions across data centers. Vnet-to-VNet connectivity is not required when configuring AlwaysON Availability Group, since all the participating VMs reside in the same Azure data centers. The VPN connectivity section can be skipped in this case.

2. Create a New Active Directory Domain Controller or Install a replica Active Directory Domain Controller or set up Azure Active Directory Sync (DirSync) in Azure Virtual Network. A domain controller is required since all the nodes of the cluster should be part of the same domain. You can create a new domain controller, install a replica of the on-premise domain controller, or set up DirSync. If you set up DirSync you can replicate all the users, OUs, and security permissions from on-premise on Azure.

3. Ensure all participating VMs in the availability group are part of the same domain. This is a prerequisite for setting up Windows Server Failover Cluster (WSFC). Set up Windows Server Failover Cluster between the participating Azure VMs or servers. There is one important caveat here which is different from setting up WSFC in an on-premise environment. After you create a WSFC between the two Azure VMs, the cluster name resource may fail to start and will not come online because it cannot acquire a unique virtual IP address from the DHCP service. Instead, the IP address assigned to the cluster name is a duplicate address from one of the nodes. This has a cascading effect that ultimately causes the cluster quorum to fail because the nodes cannot properly connect to one another.

4. To overcome this, you can assign an unused static IP address from the VNet or link local address 169.254.1.1 (for more information on link local address, refer to http://www.ietf.org/rfc/rfc3927.txt) to the cluster IP resource and bring the cluster name resource online. To create a WSFC between Azure VMs you need to use the following procedure:

    a. Create a one-node cluster on one of the Windows Azure VMs.

    b. Set the cluster IP address to static IP in VNet or IP address 169.254.1.1, which is a link local address.

    c. Bring the "Cluster Name" resource online so that it will create an entry in the active directory.

    The PowerShell script to create a one-node cluster with link local address can be found here:

https://gallery.technet.microsoft.com/scriptcenter/Create-WSFC-Cluster-for-7c207d3a

5. Add the other nodes to the cluster.

6. Ensure all the Windows Clustering Hotfixes are applied so that WSFC functions correctly.

7. Install standalone instances of SQL Server in the participating VMs. You can also provision a SQL Server image from the gallery at the time of provisioning of the VM as discussed earlier. This will pre-install a standalone instance of SQL Server on the VMs.

8. Open the firewall ports for AlwaysON data replication traffic between the VMs. Similar to Database Mirroring, AlwaysON Availability Groups communicates on TCP/IP port 5022 by default. In high-availability configuration, when all the availability replica VMs are running behind the same cloud service and location, there is no requirement to create endpoints on Azure VM. However for DR, HADR, or Hybrid IT scenarios in which replicas are spread across data centers or locations, we need to create endpoints in Azure VMs participating in the availability group to open up TCP/IP port 5022 for mirroring traffic.

Preparing SQL Environment for AlwaysON Availability Group

1. All the SQL Server services in Azure VMs should be running under a domain account. The service accounts need not be the same and can be running under different domain accounts as per the applicable security practice, but each should be a domain account if possible.

2. Enable AlwaysON Availability Group Feature in each SQL instance participating in the availability group using SQL Configuration Manager, as shown in following figure:

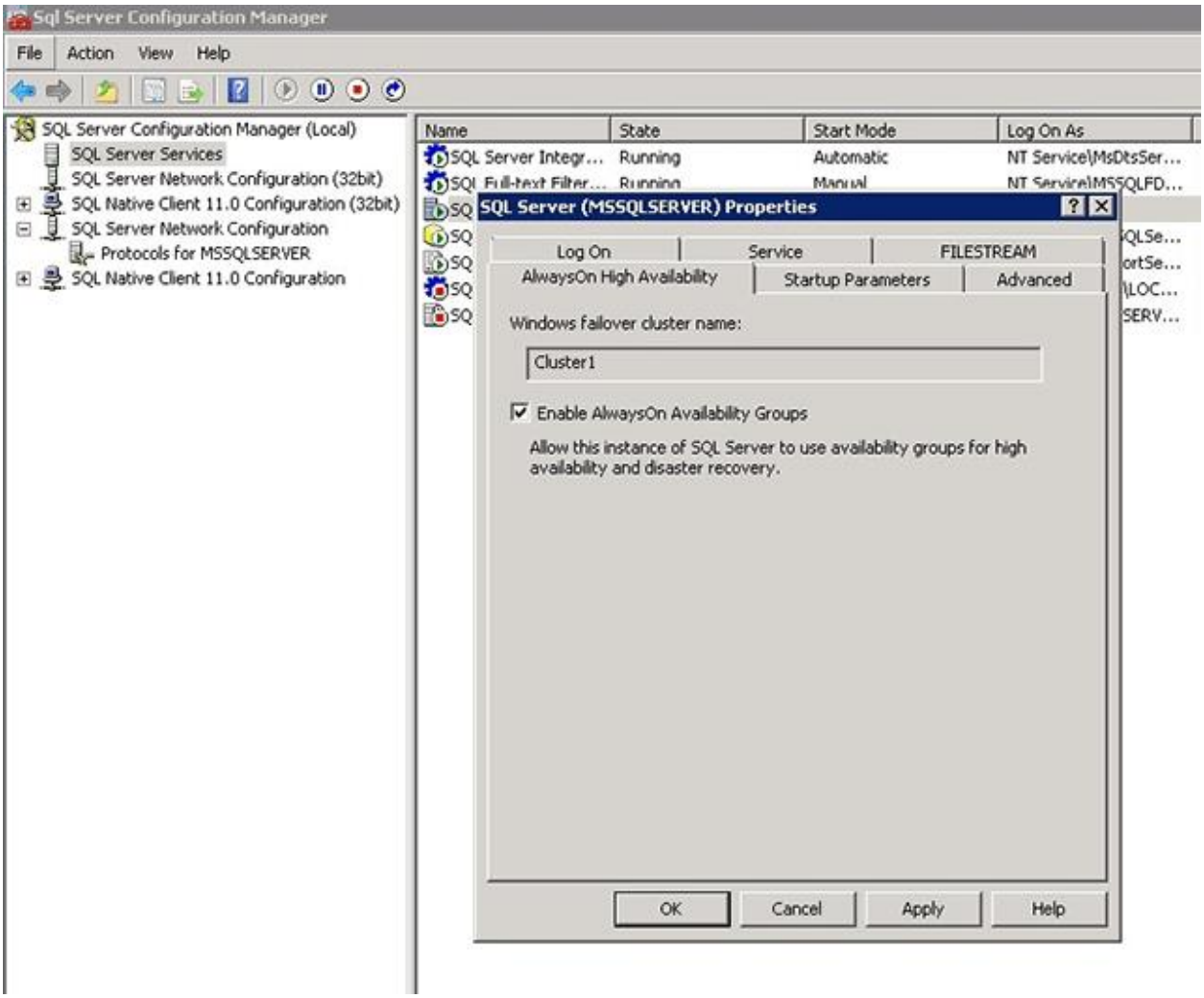

Figure 9: Enable AlwaysON AG Feature from SQL Configuration Manager

If there are any errors, it means WSFC is not configured correctly since the AlwaysON AG feature of the SQL instance checks if the server is part of WSFC under a domain and; if the feature detects any inappropriate configuration of WSFC, it fails to enable the AlwaysON AG feature.

3. Finally, the account with which you have logged in to create availability groups must have sysadmin privileges on all the SQL instance replicas to be able to create availability groups.

Creating Availability Groups

Once the prerequisites are met, you are now ready to create availability groups on SQL instances. Below are the steps you need to perform to create availability groups:

1. Ensure all the databases which need to be added as part of the availability group are created by the application on the primary replica and are set to Full Recovery Model.

2. Create a full database backup followed by a transaction log backup of all the databases which need to be added to the availability group.

3. Restore the full database backup and T-log backup created in step 2 on all the secondary replicas with the NORECOVERY option.

4. Create an AlwaysON Availability Group using the Availability Group Wizard in Management Studio (SSMS). Specify the availability group name and select the databases you would like to add to the availability group. Add the secondary replica SQL instances where you have already restored the database and transaction log with NORECOVERY option. While adding the secondary replica, you can specify whether each of the secondary replicas will be in synchronous-commit or asynchronous-commit mode and whether it will be an automatic failover partner or not. **DO NOT CREATE** the listener at this step since running AlwaysON Listener in Azure requires us to first set up Azure Internal Load Balancer (ILB). Click Join-only to join the replicas and complete the wizard to create an AlwaysON Availability Group.

5. Once the AlwaysON Availability Group is created, you can connect to Object Explorer. In SSMS, expand AlwaysOn High Availability, then expand Availability Groups; you should now see the new availability group in this container. Right-click the availability group name and click **Show Dashboard**. This dashboard shows the current states of all the replicas (Synchronized or Synchronizing). If you would like to test the failover, you can click on the Start Failover Wizard link in the dashboard to initiate the failover to secondary replicas.

Note: Do not try to fail over the availability group from the Failover Cluster Manager. All failover operations should be performed from within the AlwaysOn Dashboard in SSMS. For more information, see Restrictions on Using The WSFC Failover Cluster Manager with Availability Groups.

Configure an ILB Listener for AlwaysON Availability Group in Azure

Availability group listeners are supported on Azure VMs running Windows Server 2008 R2, Windows Server 2012, and Windows Server 2012 R2. This support is made possible by the use of Azure Internal Load Balancer (ILB) and load-balanced endpoints with direct server return (DSR) enabled on the Azure VMs that are also availability group nodes. Only one availability group listener is supported per cloud service because the listener is configured to use either the cloud service VIP address or the VIP address of the ILB. Note that this limitation is still in effect, although Azure now supports the creation of multiple VIP addresses in a given cloud service. Microsoft has more information on how to configure an ILB listener for AlwaysOn Availability Groups in Azure.

Client Configuration for Connecting to AlwaysON Azure ILB Listener

For the client application to be able to connect to the AlwaysOn Azure ILB Listener, it must reside on a different cloud service than the one that contains your availability group VMs. Azure does not support direct server return with client and server in the same cloud service.

You must follow special configuration steps for the listeners to work for both client applications that are running in Azure as well as those running on-premise. Clients must connect to the listener from a machine that is not in the same cloud service as the AlwaysOn Availability Group nodes. If the availability group spans multiple Azure subnets, such as in DR, HADR, or Hybrid scenarios, the client connection string must include "MultisubnetFailover=True" to be able to connect to IP addresses from multiple VNets quickly to support application transparent failover. This results in parallel connection attempts to the replicas in the different subnets.

You can still connect to each availability replica separately by connecting directly to the service instance. Also, since AlwaysOn Availability Groups are backward compatible with database mirroring clients, you can connect to the availability replicas, such as database mirroring partners, as long as the replicas are configured in a manner similar to database mirroring: one primary replica and one secondary replica, with the secondary replica configured as non-readable (**Readable Secondary** option set to **No**)

An example client connection string that corresponds to this database mirroring-like configuration using ADO.NET or SQL Server Native Client is below:

Code Listing 20: Client Connection String to Connect to AlwaysON AG as Mirroring Failover Partner

```
Data Source=ReplicaServer1;Failover Partner=ReplicaServer2;Initial
Catalog=AvailabilityDatabase;
```

You can use the above connection string as well if you do not wish to set up AlwaysON ILB Listener but would still like to use AlwaysON Availability Groups as an HADR solution in Azure. If you do not configure Azure ILB Listener, the only functionality which you will lose is application transparent failover wherein if the primary replica fails and the secondary replica instance becomes the new primary replica, the client connection might not connect transparently to the new primary replica and might need manual intervention to change the connection string. In this scenario, you can use client alias to ensure application config files do not need to be changed time and again as failover occurs and only alias needs to be quickly pointed to the primary replica.

Backup and Restore with Azure Blob Storage Service

Backup and restore with Azure Blob Storage Service is supported by SQL Server starting with SQL 2012 SP1 CU2. This method can be used as a disaster recovery solution only for SQL instances running SQL 2012 and later. This feature can be used to back up SQL Server databases residing in an on-premise instance or an instance of SQL Server running on Azure VMs to create off-site backup copies. Backing up to the cloud offers benefits such as availability, geo-replicated off-site storage, and ease of migration of data to and from the cloud. Backing up to Azure Blob Storage is supported using T-SQL, SMO, and PowerShell as well as starting with SQL 2014 SSMS.

Backup to Azure Blob Storage can be implemented as a disaster recovery solution in the following two topologies depending on whether the database is backed up from an on-premise SQL instance or from a SQL server running on Azure VMs.

Disaster Recovery Solution for SQL Server in Azure

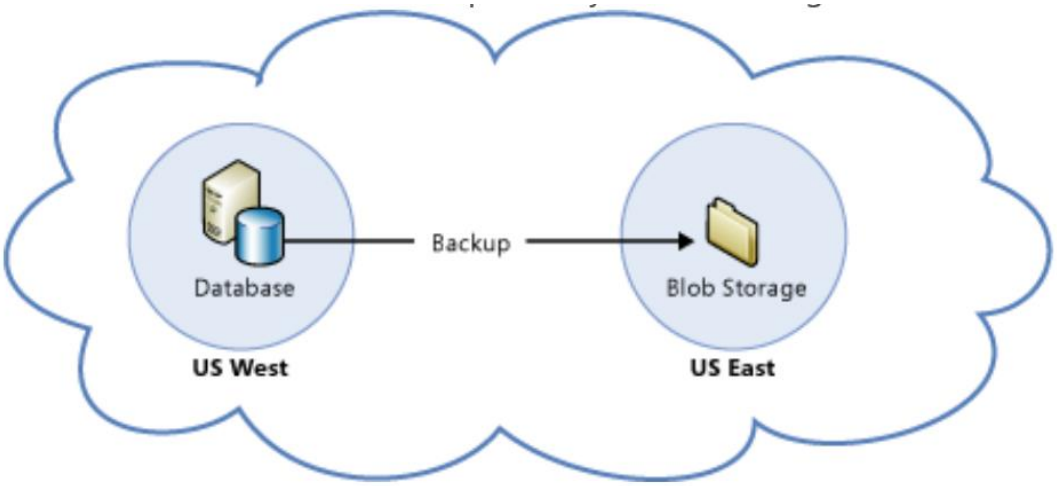

Figure 10: Backup to Azure Blob Storage as a DR Solution for SQL Server on Azure VMs

Hybrid IT: Disaster Recovery Solution for On-Premise SQL Servers in Azure

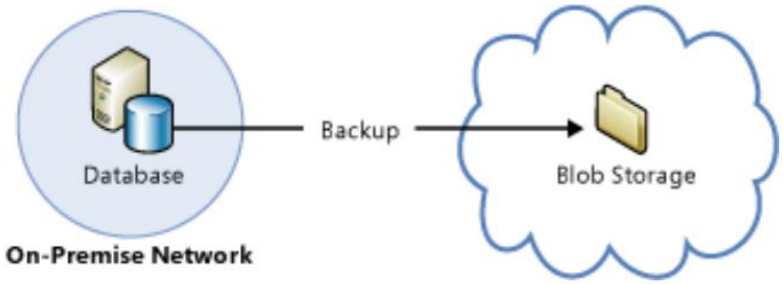

Figure 11: Hybrid IT: Backup to Azure Blob Storage as a DR Solution for On-Premise SQL Server Instances

SQL Server Backup to Azure Blob Storage

As discussed earlier, when backing up to Azure Blob storage it is recommended to turn on BACKUP compression, which was introduced to SQL Server starting with SQL 2008 R2, in order to reduce storage cost and save network bandwidth. The following article provides the detailed steps to set up SQL Server backups to Azure Blob storage:

https://msdn.microsoft.com/en-us/library/dn435916(v=sql.120).aspx

Chapter 6 Introduction to Azure SQL Database (PaaS)

Azure SQL Database is the new cloud-based, database as a service offering from Microsoft. It allows you to spin up a database in Azure within minutes without worrying about the VM size, OS, or SQL installation. Azure SQL Database is suitable for developers or DevOps designing new applications who do not need to worry about installation, configuration, and management of SQL Server instances and would like to focus on the functionality and features provided by the relational database.

Azure SQL Database allows users to scale up single databases with no downtime and provides predictable performance that can be dialed up or down as needed. You can also choose elastic databases to scale out to thousands of databases within a budget that you control, leveraging the powerful manageability features of elastic database jobs and the programmatic features of elastic database tools.

Azure SQL Database provides an SLA of 99.99% (four nines), which translates to a downtime of 52.56 minutes per year. The built-in data protection via auditing, restore, and geo-replication helps ensure your data is protected. SQL Database provides you with the ability to restore your data to any point in time from any transaction for up to 35 days. You can also replicate your data to an Azure region of your choice and easily implement a geographic disaster recovery policy designed for your business needs. The latest version of SQL Database, V12, provides nearly complete compatibility with the SQL Server on-premise engine (box version).

Azure SQL Database is available in Basic, Standard, and Premium service tiers, which support light-weight to heavy-weight database workloads, so you can move across or blend the tiers together for innovative application designs. With the power and reach of Azure, you can mix and match Azure services with SQL Database to meet your unique modern app design needs, drive cost and resource efficiencies, and unlock new business opportunities.

Azure SQL Database Service Tiers

As mentioned earlier, Azure SQL Database is available in three service tiers: Basic, Standard, and Premium. Each service tier provides an increasingly higher amount of capacity and processing power measured in terms of Database Transaction Units (DTUs). To understand service tiers, it is important to first understand Database Transaction Units (DTUs), a new unit of transactional performance coined in Azure SQL Database Benchmark (ASDB).

Database Transaction Units (DTUs)

DTUs are not a real world metric like CPU utilization or transactions per second, but they are a complex calculation. They are coined as a measure or unit of relative capacity of a database performance level provided by each service tier based on a blended measure of CPU, memory, and read and write rates. Doubling the DTU rating of a database equates to doubling the database power. The reason for introducing this unit was to provide a relative measure of transactional processing power of the SQL Database across different service tiers. Just as IOP is to storage, and frequency is to CPUs, DTUs are to Azure SQL Database.

The Microsoft Product Team performed a benchmark test on Azure SQL Database to try to produce a mix of database operations (insert, update, delete, and select) common in all OLTP workloads against a schema containing a range of tables and data types. The goal of the benchmark is to provide a reasonable guide to the relative performance of a database that might be expected when scaling up or down between performance levels. The detailed overview of the Azure SQL Database Benchmark (ASDB) can be found in the following article:

https://azure.microsoft.com/en-us/documentation/articles/sql-database-benchmark-overview/

Now that you understand Database Transaction Units, let us compare the various service tiers of Azure SQL Database below.

Basic Tier

The Basic tier is designed for applications with very low transactional activity. It might be the preferred tier when a developer is developing the application with only a single user connected to and querying on the database.

A typical use case is a lightweight application that needs a small database with a single operation at any given point in time.

Table 4: Performance Size and Features Supported by Basic Tier

Service Tier	Basic
Database Throughput Units (DTUs)	5
Maximum Database Size	2 GB
Point-in-time Restore (PITR)	Up to millisecond within last 7 days
Disaster Recovery	Geo-Restore, restore to any Azure region

Service Tier	Basic
Performance Objectives	Transaction rate per hour

Standard Tier

The Standard tier offers better performance and built-in business continuity features compared to the Basic tier, which can support multiple concurrent transactions. A typical use case is an application with moderate concurrency that needs a medium-sized database.

Table 5: Performance and Size Supported by Standard Tier

Service Tier	Standard S0	Standard S1	Standard S2	Standard S3
Database Throughput Units (DTUs)	10	20	50	100
Maximum Database Size	250 GB	250 GB	250 GB	250 GB

Table 6: Features Supported by the Standard Tier

Service Tier	Standard (S0, S1, S2, S3)
Point-in-time Restore (PITR)	Up to millisecond within last 14 days
Disaster Recovery	Standard geo-replication, 1 offline secondary
Performance Objectives	Transaction rate per minute

Premium Tier

The Premium tier offers the best level of performance and advanced business continuity features with active geo-replication in up to four Azure regions of your choice. A typical use case is a mission-critical application with high transactional volume and many concurrent users.

Table 7: Performance and Size Supported by Premium Tier

Service Tier	Premium P1	Premium P2	Premium P4	Premium P6 (formerly P3)
Database Throughput Units (DTUs)	125	250	500	1000
Maximum Database Size	500 GB	500 GB	500 GB	500 GB

Table 8: Features Supported by Premium Tier

Service Tier	Premium (P1, P2, P4, P6)
Point-in-time Restore (PITR)	Up to millisecond within last 35 days
Disaster Recovery	Active geo-replication, up to 4 online readable secondaries
Performance Objectives	Transaction rate per second

Note: The above specs for the service tiers are the numbers supported by Azure SQL Database as of today. As Azure SQL Database develops and grows, the specs or service tier may also grow. The latest details on the service tier can be found in the

following article: ***https://azure.microsoft.com/en-us/documentation/articles/sql-database-service-tiers/***

Azure Database Resource Limits

Besides the size and transactional limitation imposed by service tiers on Azure SQL Database, there are certain additional resource restrictions on the Azure SQL Database to ensure badly written code doesn't disrupt the performance of the server on which the database is deployed, as Azure is a shared multi-tenant environment.

Table 9: Resource Limits on Azure SQL Database

Resource	Default Limit
Database size	Depends on Service Tier
Logins	Depends on Service Tier
Memory usage	16 MB memory grant for more than 20 seconds
Sessions	Depends on Service Tier
Tempdb size	5 GB
Transaction duration	24 hours
Locks per transaction	1 million
Size per transaction	2 GB
Percent of total log space used per transaction	20%
Max concurrent requests (worker threads)	Depends on Service Tier

The error codes received when you hit any of the above limitations and the latest resource limits are documented in the following article:

https://azure.microsoft.com/en-us/documentation/articles/sql-database-resource-limits/

What's new in Azure SQL Database V12

When Azure SQL Database was first introduced, one of the major drawbacks was its lack of parity with the on-premise version of SQL Server. It appeared as if Microsoft took a step backwards with SQL Server with Azure SQL Database, but it may be due to the fact the Azure SQL Database code was rewritten to make it a cloud service. The latest version of Azure SQL Database is at par with SQL 2014, and in fact some of the latest service updates have exceeded the features provided by SQL 2014 and are due to come in the on-premise version of SQL 2016.

The following enhancements can be seen in Azure SQL Database v12.

T-SQL Programming and Application Compatibility

A key goal for SQL Database V12 was to improve compatibility with Microsoft SQL Server 2014. Among other areas, V12 achieves parity with SQL Server in the important area of programmability. For instance:

- Common Language Runtime (CLR) assemblies
- Window functions, with OVER
- XML indexes and selective XML indexes
- Change tracking
- SELECT...INTO
- Full-text search

However there are still T-SQL Features which are either partially supported or not supported in Azure SQL Database, all of which are documented in the following MSDN article:

https://azure.microsoft.com/en-us/documentation/articles/sql-database-transact-sql-information/

Performance Enhancements

In V12, the DTUs allocated to all Premium performance levels was increased by 25% and new performance-related features were added, including:

- In-Memory Columnstore Index

- Table Partitioning by rows
- Dynamic Management Views (DMVs) and Extended Events (XEvents) to help monitor and tune performance

Security Enhancements

Security is a primary concern for anyone who runs their business in the cloud. The latest security features released in V12 include:

- Row-level security (RLS)
- Dynamic Data Masking
- Contained databases
- Application roles managed with GRANT, DENY, REVOKE
- Transparent Data Encryption (TDE)

Business Continuity Enhancements

V12 made some major improvements in recovery point objectives (RPOs) and estimated recovery times (ERTs) for geo-restore and geo-replication, which gives you the confidence for running mission critical applications on Azure SQL Database. The following table lists the RPO and ERTs enhancements in V12.

Table 10: Business Continuity Enhancements in V12

Business continuity feature	Earlier version	V12
Geo-restore	• RPO < 24 hours • ERT < 12 hours	• RPO < 1 hour • ERT < 12 hours
Standard geo-replication	• RPO < 30 minutes • ERT < 2 hours	• RPO < 5 seconds • ERT < 30 seconds
Active geo-replication	• RPO < 5 minutes • ERT < 1 hour	• RPO < 5 seconds • ERT < 30 seconds

Introducing Elastic Database Pool

Elastic Database Pool allows you to programmatically create, maintain, and manage performance (including DTUs) across a group of databases within the desired spending limit. This feature is specifically beneficial for cloud SaaS (Software as a Service) developers who like to scale out databases and would prefer to leverage the shared resources in the pool to overcome some of the performance and sizing restrictions imposed by a service tier.

At the time of writing, this feature is still in preview and will go live soon, but you can track the service updates for features with Azure SQL Database by subscribing to the following feed:

http://azure.microsoft.com/en-us/updates/?service=sql-database&update-type=general-availability

Developing with Azure SQL Database

Azure SQL Database supports connectivity from clients running on Windows, Mac OS, and Linux. Microsoft also provides SDKs and samples for developing on Azure SQL Database for a variety of programming languages namely .NET, Java, PHP, Python, and Ruby. You can find short, quick-start samples to connect, query, and retry to Azure SQL Database from various programming languages in the following article:

https://azure.microsoft.com/en-us/documentation/articles/sql-database-develop-quick-start-client-code-samples/

Chapter 7 Provisioning, Managing, and Migrating to Azure SQL Database

One of the best parts of Azure SQL Database is that Microsoft has integrated it with all the familiar tools like Azure Portal, PowerShell, and SQL Server Management Studio. The learning or ramp-up curve for provisioning, managing, and migrating to Azure SQL Database is not that steep and is very natural.

Provisioning an Azure SQL Database

There are two ways to provision an Azure SQL Database in an Azure Subscription:

- Using Azure Management Portal
- Using Azure PowerShell

Azure Management Portal can be used as a GUI interface for provisioning the database while Azure PowerShell is useful as a programmatic interface for automation.

To provision an Azure SQL Database in Azure, you must first obtain an Azure subscription. You can purchase an Azure subscription or sign up for an Azure free trial subscription.

Using Azure Management Portal

After you obtain your Azure subscription, you can follow the steps in this section to provision an Azure SQL Database using Azure Management Portal.

1. Log in to Azure Management Portal and Click on New > Data + Storage > SQL Database as shown below

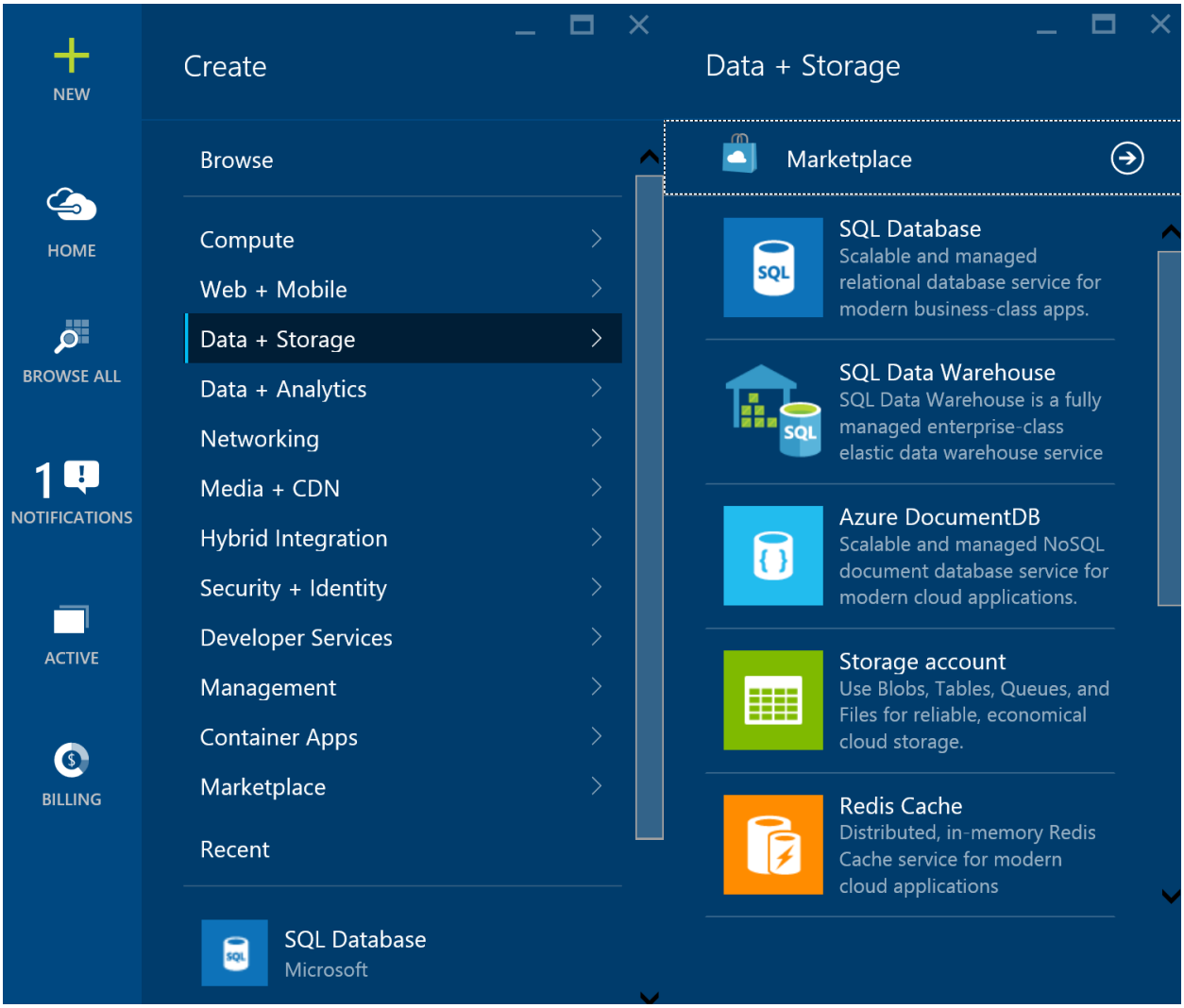

Figure 12: Provisioning an Azure SQL Database

2. The next step is to provide the name of the database and the logical SQL server name where you would like to host the database. The database name should be unique within a given server and is checked by the portal as you are typing. You can use an existing server name if you have already created it earlier or create a new server as shown in following figure.

   You can think of the logical server as a SQL instance provisioned in Azure that will be hosting your Azure SQL database. While creating a server, you need to provide a unique server name with a DNS suffix of *database.windows.net*, login name and password of sysadmin on that server, and Azure location where you want to provision this database to be hosted. This is shown in the following figure:

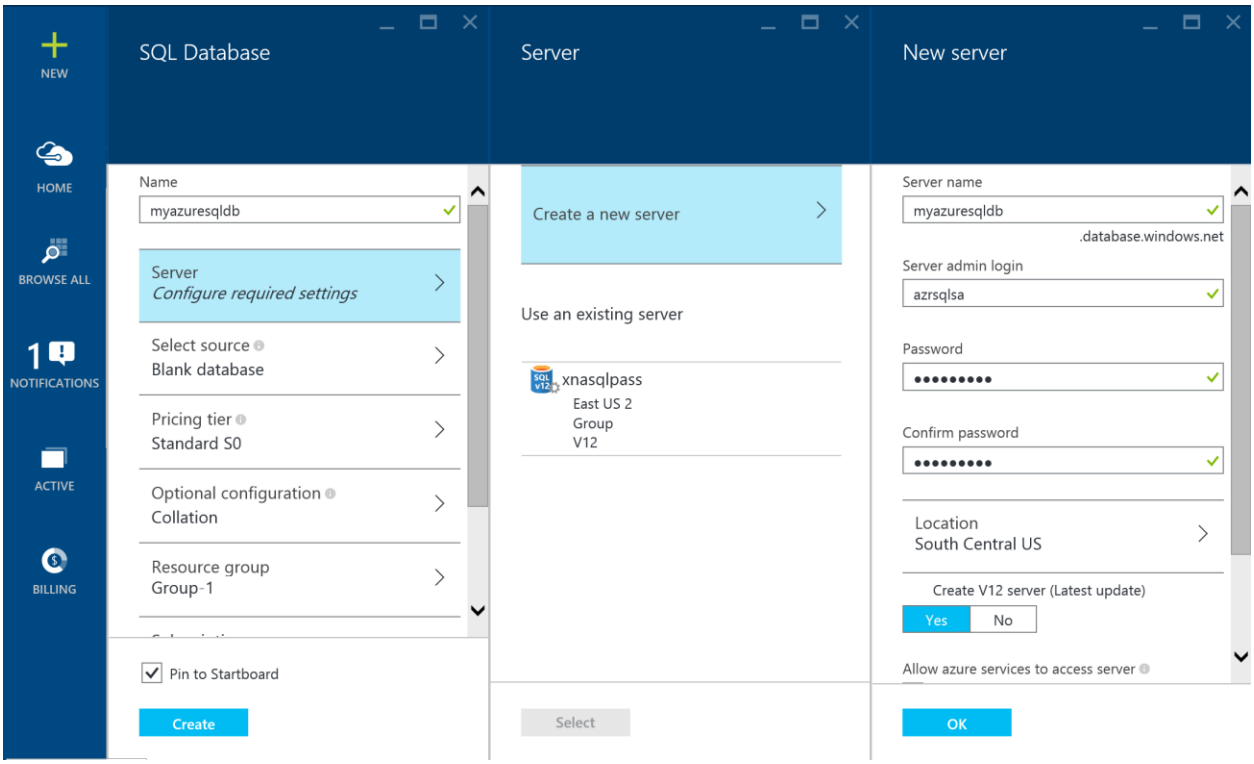

Figure 13: Creating a New Server for Azure SQL Database

3. The next step is to select the source for the database. You can create a blank database and create schema/objects or select a sample database (yes, our best friend Adventureworks is available as a sample database in Azure SQL database as well) or restore a database from a backup.

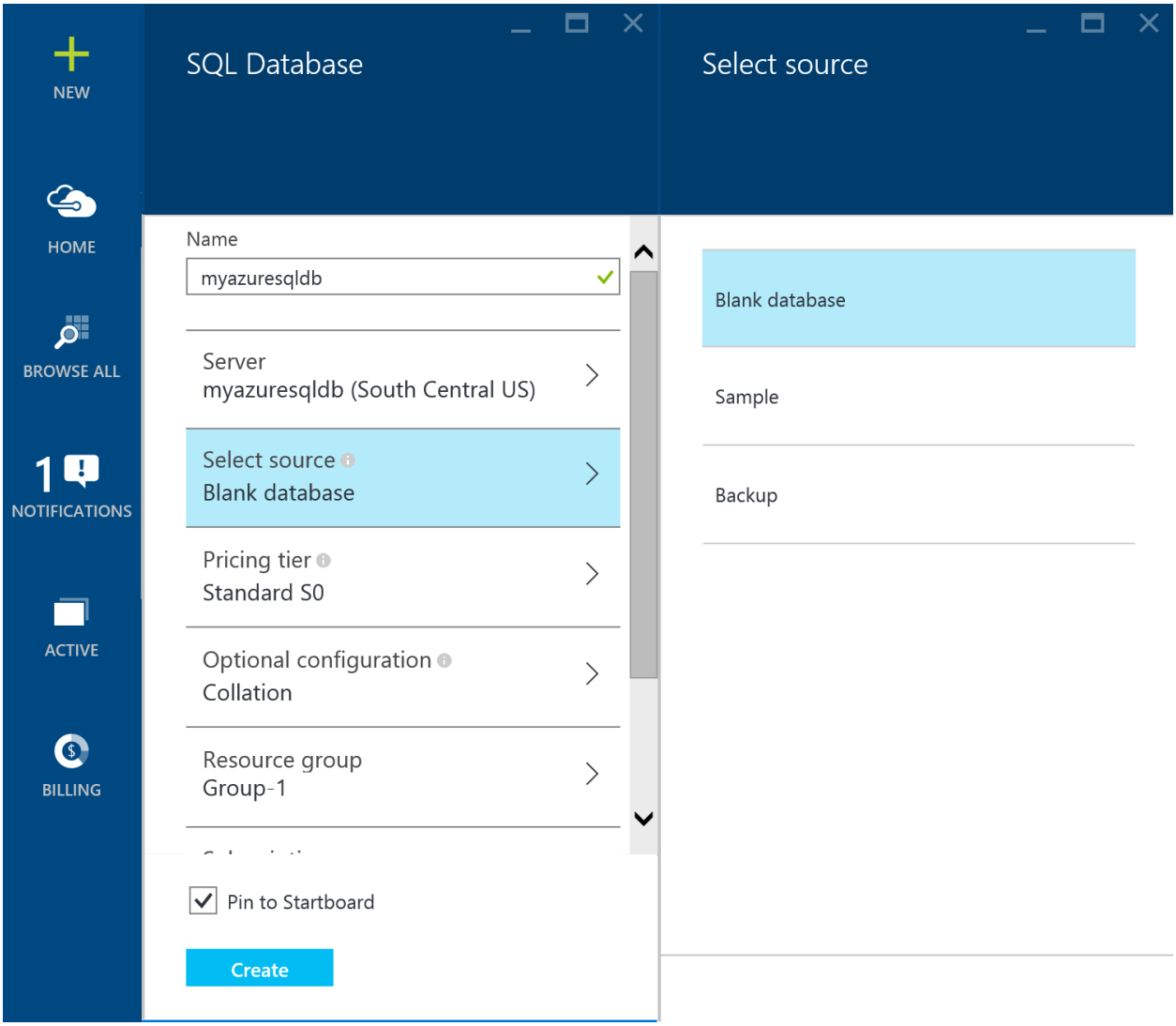

Figure 14: Selecting the Source for Azure SQL Database

4. The next step is to select the service tier for the database. The service tier also decides the per-minute pricing of the database and is referred to as Pricing Tier in the portal.

5. Next, select the desired collation for the database and the resource group to which the database belongs. Resource Group is a new concept in Azure to group and manage the lifecycle and dependencies of applications running on Azure. If the Azure SQL Database is created as a backend for running an Azure service application, you can categorize all the resources belonging to that application in a single resource group which can be managed and monitored as a single unit. For a sample database, you can choose the Default-SQL-<Location> Resource Group available in the list.

6. Select the Azure subscription under which the database will be created and billed. This is useful when you manage multiple Azure subscriptions.

7. Finally, click Create at the bottom of the window for the Azure SQL Database to be provisioned in the specified logical server.

Using Azure PowerShell

Using Azure PowerShell, you can provision an Azure SQL Database using the following PowerShell code:

Code Listing 21: Provisioning an Azure SQL Database

```
Import-Module Azure

Add-AzureAccount


#Select the desired Azure Subscription

[array] $AllSubs = Get-AzureSubscription

If ($AllSubs)

{

        Write-Host "`tSuccess"

}

Else

{

        Write-Host "`tNo subscriptions found. Exiting." -ForegroundColor
Red

        Exit

}

#Write-Host "`n[Select Option] - Select a Subscription to Work With" -
ForegroundColor Yellow

$count = 1

ForEach ($Sub in $AllSubs)

{

   $SubName = $Sub.SubscriptionName

   Write-host "`n$count - $SubName"  -ForegroundColor Green

   $count = $count+1

}

$SubscriptionNumber = Read-Host "`n[SELECTION] - Select a Subscription to
Provision the VM"
```

```
If($SubscriptionNumber -gt $count)
{
    Write-Host "`Invalid Subscription Entry - Existing" -ForegroundColor
Red
    Exit
}
$SelectedSub = Get-AzureSubscription -SubscriptionName
$AllSubs[$SubscriptionNumber - 1].SubscriptionName 3>$null
$SubName = $SelectedSub.SubscriptionName
Write-Host "`n Selected Subscription - $SubName" -ForegroundColor Green
$SelectedSub | Select-AzureSubscription | out-Null


$Location = "South Central US"
$DatabaseName = "myazuresqldb"


$server = New-AzureSqlDatabaseServer  -Location $location –
AdministratorLogin "mysqlsa" -AdministratorLoginPassword "Pa$$w0rd" -
Version "12.0"



New-AzureSqlDatabase  -ServerName $server.ServerName -DatabaseName
$DatabaseName -Edition "Basic" -MaxSizeGB 2
```

Managing Azure SQL Database

As mentioned earlier, one of the best parts of Azure SQL Database is that Microsoft has maintained the same tools, namely SQL Server Management Studio, TSQL, and PowerShell, for managing Azure SQL Database. This makes it easier to manage due to the familiarity of the tools.

In order to be able to manage and connect to Azure SQL Database, you need to first create firewall rules to allow connections to your servers and databases from outside. You can define server-level and database-level firewall settings for the master or a user database in your Azure SQL Database server to selectively allow access to the database.

Configure Server-Level Firewall Rules

Server-level firewall rules can be created and managed through the Microsoft Azure Management Portal, Transact-SQL, and Azure PowerShell.

The following steps are used to create server-level firewall rules using Azure Management Portal:

1. Log in to Azure Management Portal with admin privileges, click on Browse All on the left banner, and click SQL Servers.

2. In the server window, click Settings at the top, and then click Firewall to open the Firewall Settings sub-window for the server.

3. Add or change a firewall rule.

    - To add the IP address of the current computer, click Add Client IP at the top of the sub-window.

    - To add additional IP addresses, type in the RULE NAME, START IP address, and END IP address.

    - To modify an existing rule, click and change any of the fields in the rule.

    - To delete an existing rule, click the rule, click the ellipsis (…) at the end of the row, and then click Delete.

4. Click Save at the top of the Firewall Settings sub-window to save the changes.

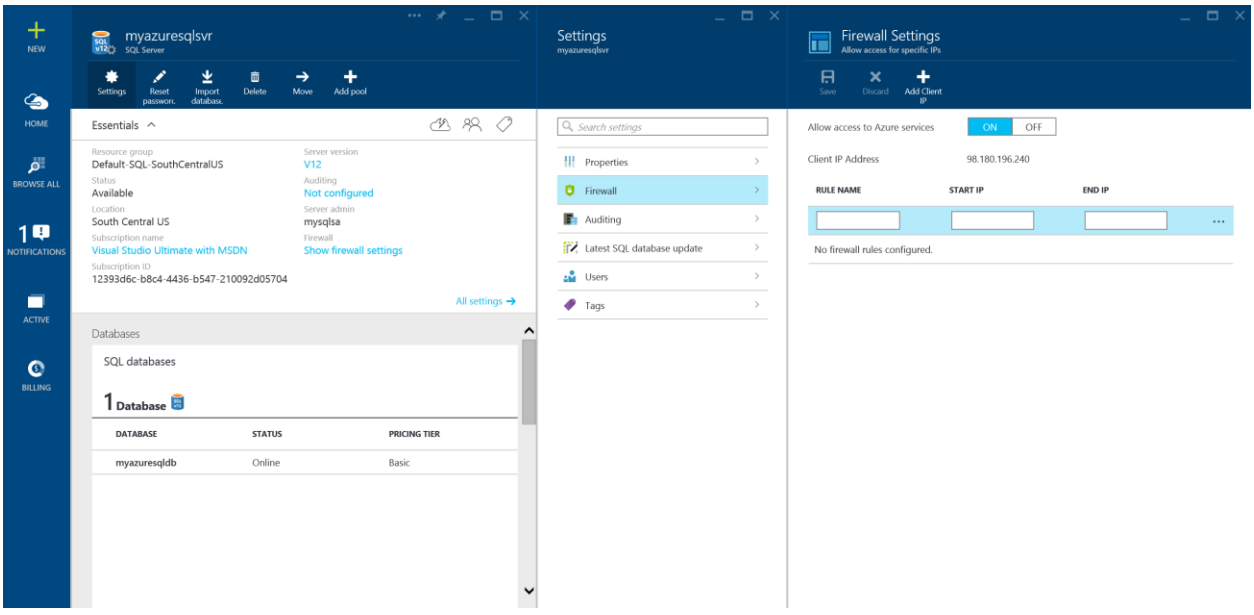

Figure 15: Configuring the Server-Level Firewall using Azure Portal

You can also configure the server-level firewall using Azure PowerShell using the following cmdlets:

Code Listing 22: Configuring Server Level Firewall using Azure PowerShell

```
Import-Module Azure

Add-AzureAccount

## To Add a new server-level firewall rule

New-AzureSqlDatabaseServerFirewallRule -StartIPAddress 172.16.1.1 -
EndIPAddress 172.16.1.10 -RuleName DBAFirewallRule -ServerName
myazuresqldbserver

## To modify an existing server-level firewall rule

Set-AzureSqlDatabaseServerFirewallRule -StartIPAddress 172.16.1.4 -
EndIPAddress 172.16.1.10 -RuleName DBAFirewallRule -ServerName
myazuresqldbserver

# To delete an existing server-level firewall rule

Remove-AzureSqlDatabaseServerFirewallRule -RuleName DBAFirewallRule -
ServerName myazuresqldbserver
```

In order to connect to Azure SQL Database Server using SQL Server Management Studio (SSMS) from a client workstation, you would first need to create the server-level firewall rule using one of the above techniques to allow the client workstation IP address as a legitimate entry to SQL Server. However, after you establish the connection to SQL Server using SSMS, you can create additional firewall rules using T-SQL queries fired against the server using Management Studio.

To create or update server-level firewall rules, execute the sp_set_firewall rule stored procedure. To delete the firewall rules, execute the sp_delete_firewall_rule stored procedure as shown below.

Code Listing 23: Configuring Server-Level Firewall Using Transact SQL

```
--Create new firewall rule

EXEC sp_set_firewall_rule @name = N'DBAFirewallRule', @start_ip_address =
'172.16.1.1', @end_ip_address = '172.16.1.10'

-- Update an existing firewall rule

EXEC sp_set_firewall_rule @name = N'DBAFirewallRule', @start_ip_address =
'172.16.1.4', @end_ip_address = '172.16.1.10'

--Delete an existing firewall rule

EXEC sp_delete_firewall_rule @name = N'DBAFirewallRule'
```

Configure Database-Level Firewall Rules

After creating server-level database rules, you will have to create a database-level firewall for the clients to be able to connect to the specific database on that server. You can configure the database-level firewall using T-SQL.

In order to create a database-level firewall, you need to connect to the server using SQL Server Management Studio from the client workstation, which is opened for connection in the server-level firewall. Then, execute the sp_set_database_firewall_rule stored procedure to create or update database-level firewall rules as shown in the following code snippet:

Code Listing 24: Configuring Database-Level Firewall Using Transact SQL

```
--Create new database-level firewall rule

EXEC sp_set_database_firewall_rule @name = N'ApplicationFirewallRule',
@start_ip_address = '172.16.1.11', @end_ip_address = '172.16.1.11'

-- Update an existing database level firewall rule

EXEC sp_set_database_firewall_rule @name = N'ApplicationFirewallRule',
@start_ip_address = '172.16.1.11', @end_ip_address = '172.16.1.12'

--Delete an existing firewall rule

EXEC sp_delete_database_firewall_rule @name = N'ApplicationFirewallRule'
```

Connecting to Azure SQL Database using SSMS

As discussed earlier, in order to connect to the Azure SQL Database server, the first step is to ensure the client workstation from where you intend to connect is added in the server-level firewall. Once the IP address is allowed to connect to the server, you can follow the steps in this section to connect to Azure SQL Database Server using SSMS for managing the server instance. SQL Server 2014 SSMS with the latest updates offers expanded support for tasks like creating and modifying Azure SQL databases. In addition, you can also use Transact-SQL statements to accomplish these tasks. The steps below provide examples of these statements. For more information about using Transact-SQL with SQL Database, including details about which commands are supported, see Transact-SQL Reference (SQL Database).

If you do not know the name of the server for a given Azure SQL database, you can identify the server from Azure Portal as described here:

1. Log in to Azure Portal, click on Browse All on the left hand banner, and click on SQL Databases.

2. Click on the Azure SQL Database you wish to connect to and you will see the server name in the database sub-window.

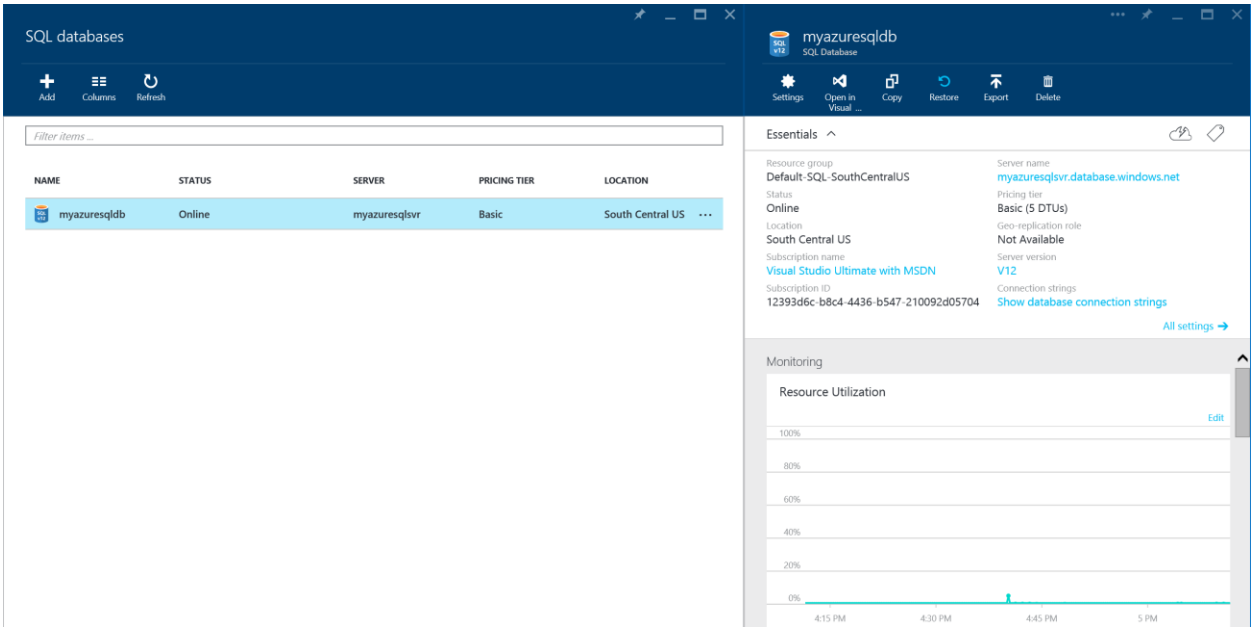

Figure 16: Server Name in Azure SQL Database

3. The server name is unique and appended with the DNS suffix *database.windows.net.*

4. On the taskbar, click Start, select All Programs, select Microsoft SQL Server 2014, and then click SQL Server Management Studio.

5. In Connect to Server, specify the fully qualified server name as *serverName.database.windows.net*, as noted earlier.

6. Select SQL Server Authentication since Windows Authentication is not yet supported in Azure SQL Database.

7. In the Login box, enter the SQL Server administrator login that you specified in the portal when you created your server.

8. In the Password box, enter the password that you specified in the portal when you created your server.

9. Click Connect to establish the connection.

Managing Azure SQL Database using T-SQL

Once connected to Azure SQL Database Server using SSMS, you can use T-SQL to create and manage database objects, as shown in the following code snippet. One important consideration while working with Azure SQL Database is that the **USE** statement is not supported for switching between databases. Instead, you need to establish or re-establish a connection directly to the target database.

Code Listing 25: Managing Azure SQL Database Using T-SQL

```
-- Create new database
CREATE DATABASE myTestDB
(
 EDITION='Standard',
 SERVICE_OBJECTIVE='S0'
);
-- ALTER DATABASE
ALTER DATABASE myTestDB
MODIFY
(
SERVICE_OBJECTIVE='S1'
);
-- DROP DATABASE
DROP DATABASE myTestDB;
-- Create New Server Login
CREATE LOGIN User1 WITH password='Password1';
-- Create Database User (switch to User Database)
CREATE USER dbuser1 FROM LOGIN User1;
--Adding User to db_datareader role (switch to User Database)
exec sp_addrolemember 'db_datareader', 'dbuser1';
-- DROP Login (Switch to master database)
DROP LOGIN User1
-- Query catalog views (on master database)
SELECT * FROM sys.databases
```

```
SELECT * FROM sys.sql_logins

-- Query dynamic management views (on User database)

SELECT text,* from sys.dm_exec_requests

cross apply sys.dm_exec_sql_text(sql_handle)

SELECT * from sys.dm_exec_connections

SELECT * from sys.dm_exec_sessions
```

Migrating to Azure SQL Database

In order to migrate an existing on-premise SQL database to Azure SQL Database, you need to first ensure the TSQL code and objects within the database are compatible with Azure SQL Database. As mentioned in previous chapters, there are still T-SQL features which are only partially supported or not supported at all in Azure SQL Database, all of which are documented in the following MSDN article:

https://azure.microsoft.com/en-us/documentation/articles/sql-database-transact-sql-information/

If a database schema is compatible with Azure SQL Database, the migration to Azure SQL Database can be done either in a single step using SSMS by deploying the database to Azure SQL Database or as a two-step process by first exporting a BACPAC of the database and then importing that BACPAC into an Azure SQL Server as a new database. We will explore each migration method below.

Note: A data-tier application (DAC) introduced in SQL 2008 R2 is an entity that contains all of the database and instance objects used by an application. A DAC provides a single unit for authoring, deploying, and managing the data-tier objects instead of having to manage them separately. A DAC allows tighter integration of data-tier development with the development of the associated application code. It also gives administrators an application-level view of resource usage in their systems. A BACPAC file is a DAC package file along with the data in the database and can be considered as a backup file for the database, which can be used to recreate the database objects and data by importing the BACPAC file.

Using SSMS to Deploy to Azure SQL Database

With the latest update in SQL 2014, SQL Server Management Studio (SSMS) comes with the Deploy Database to Azure SQL Database wizard, allowing you to directly deploy an on-premise version of SQL Database to Azure SQL Database provided its schema is compatible. The following steps demonstrate how to use the deployment wizard in SSMS to migrate the database.

1. Using SSMS 2014 SP1, connect to the on-premise database to be migrated, right-click on the database in the object explorer and click task to see the “Deploy Database to Microsoft Azure SQL Database” option as shown in the figure.

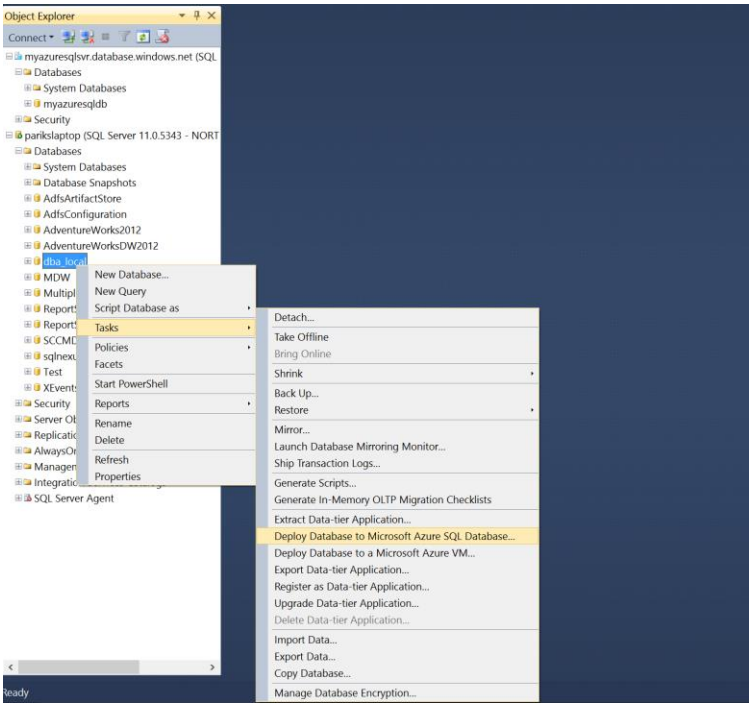

Figure 17: Deploy Database to Microsoft Azure SQL Database Wizard

2. Click Next and connect to Azure SQL Database Server using SQL authentication by providing the admin username and password. Select the database edition, service tier, and max size as shown in the following figure.

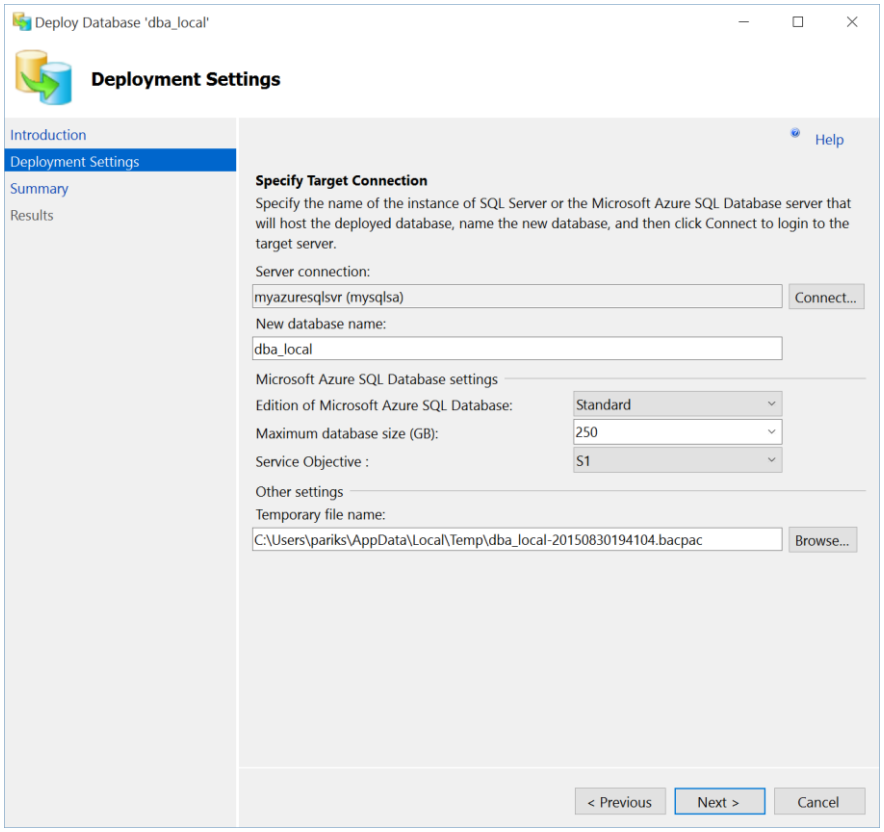

Figure 18: Deploy Database to Microsoft Azure SQL Database

3. Click Next and then Finish to let the wizard extract the database schema and export data to a BACPAC file, which is then imported into Azure SQL Database.

Depending on the size and complexity of the database, deployment may range from a few minutes to many hours. If there are incompatibilities, the schema validation routines will quickly detect errors before any deployment to Azure actually occurs. This method is useful when the entire database schema is compatible with Azure SQL Database and you want to deploy the entire database in one go. If, however, only a subset of the database schema is compatible with Azure SQL Database and you would like to selectively extract database objects to Azure SQL Database, you can manually export the BACPAC and copy it to Azure Blob storage and restore it on Azure SQL Database Server, as discussed in the next section.

Using SSMS To Export a BACPAC and Import It to SQL Database

This method can be broken into two main steps:

1. Export the database objects to a BACPAC file on local storage or Azure blob storage.
2. Import the BACPAC file to Azure SQL Database Server.

This method involves the following steps:

1. Connect to the source database in SSMS and export to a BACPAC file as shown in the following figure:

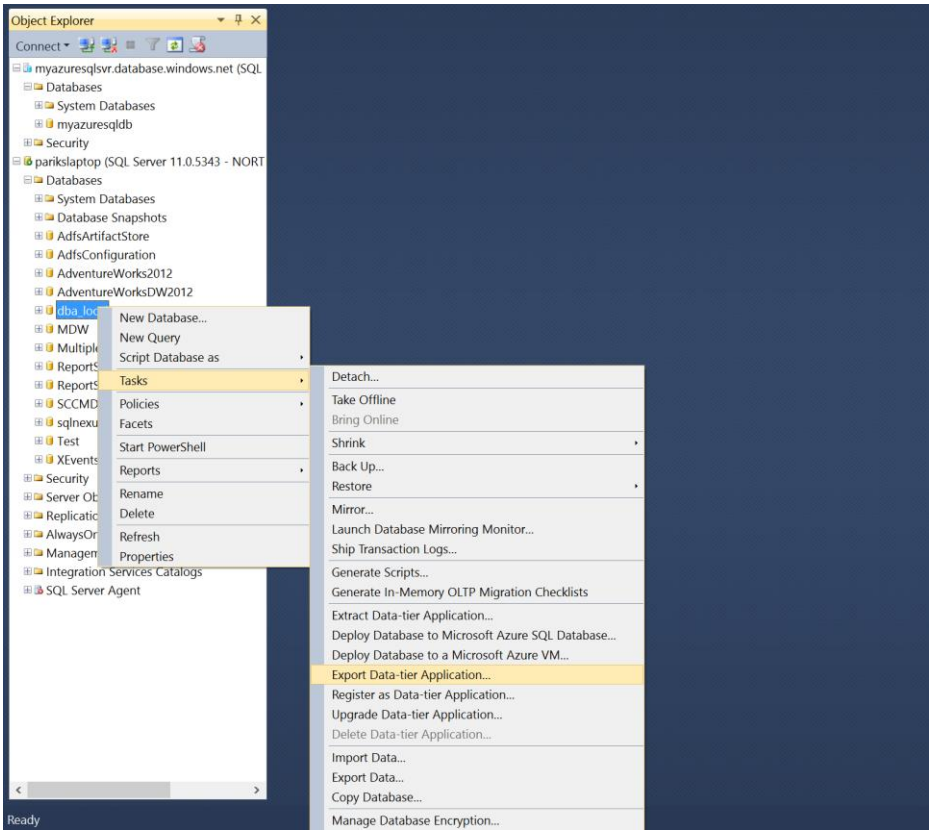

Figure 19: Exporting a SQL Database to BACPAC

2. In the export wizard, configure the export to save the BACPAC file to either a local disk location or to Azure Blob storage. Click the Advanced tab if you want to exclude data from some or all of the tables, as shown in the following figure:

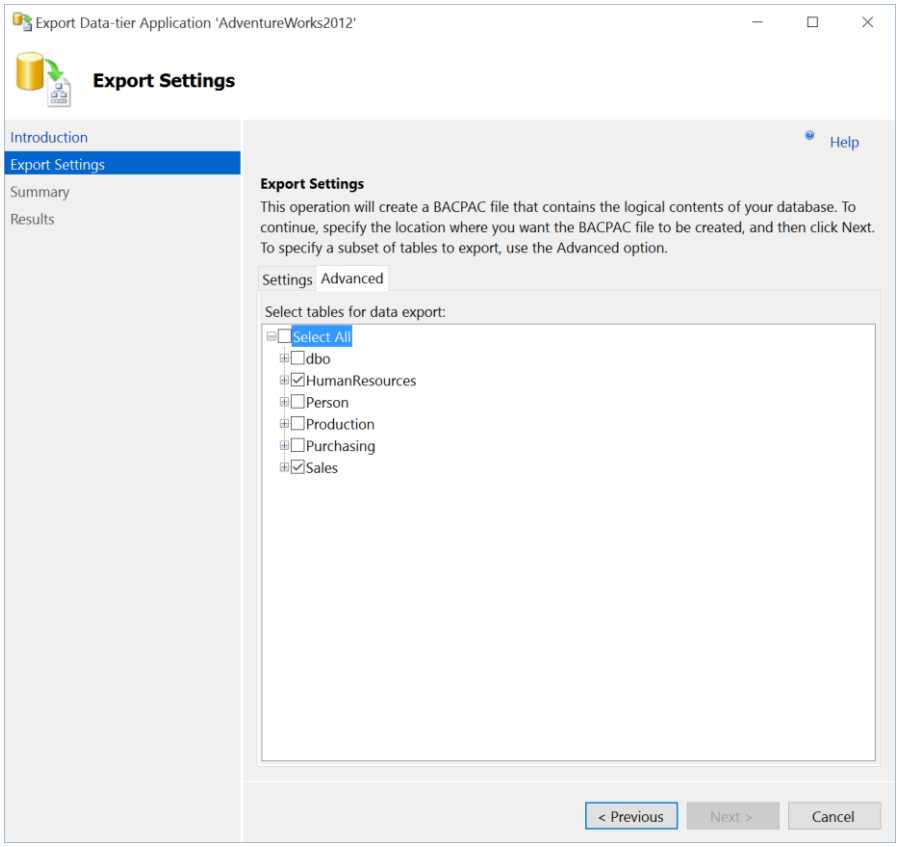

Figure 20: Export to BACPAC by Filtering the Database Objects

3. Once the BACPAC has been created, connect to Azure SQL Database Server using SSMS and right-click on the Databases folder to import the BACPAC file.

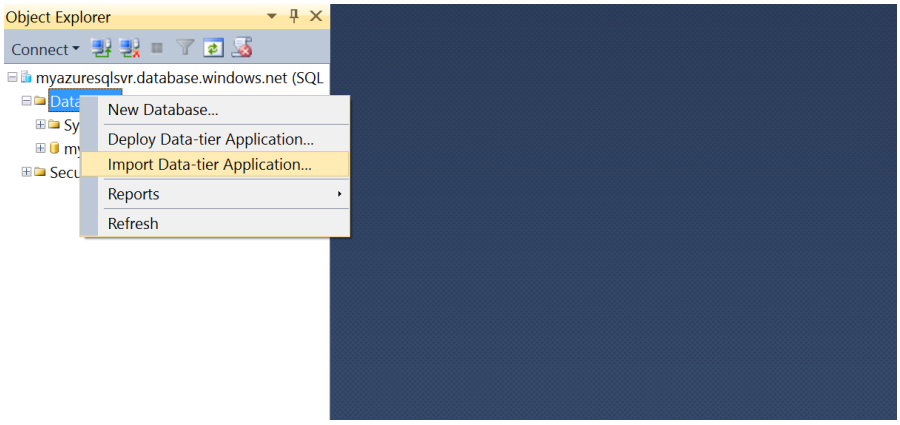

Figure 21: Import the BACPAC file in Azure SQL Database Server

4. While importing, you need to provide the edition, service tier, and max size of the database you want to create and complete the wizard to finish importing the BACPAC file to Azure SQL Database Server.

This method gives you finer control over the objects you want to migrate to Azure SQL Database and hence is useful when you only want to migrate a partial database to Azure SQL Database.

Chapter 8 Performance Considerations for Azure SQL Database

In the previous chapters you learned how easy it is to set up, provision, manage, and migrate to Azure SQL Database. Reduced setup time, ease of management, on-demand capacity scaling, reduced cost, and administrative overhead are some of the biggest motivations for using Azure SQL Database as a backend for your application. However, performance and security are two important aspects which actually drive the business decision for choosing a database platform. In this chapter, you will learn about the performance monitoring tools and considerations for running your database workload in Azure SQL Database.

Choosing the Right Service Tier for Your Workload

In an on-premise setup of SQL Server, you choose the right hardware, memory, and storage to ensure you get a predictable performance. When using SQL on Azure VM (IaaS), you choose the right VM size and storage (premium or standard) to ensure you get a predictable performance. Similarly, when using Azure SQL Database you need to choose the right service tier for your workload to ensure predictable performance for your workload. Service tiers are therefore often referred to as performance tiers for Azure SQL Database.

The following table can be useful as a decision matrix for choosing the right service tier for your workload.

Table 11: Choosing the Right Service Tier for Your Workload

	Basic	**Standard**	**Premium**
Database Size	< 2 GB	< 250 GB	< 500 GB
Concurrency	Single user	Small to medium	Medium to large
Response time	High	Medium	Low
Best for	Getting started or single user development on Azure SQL Database	Servicing websites with moderate traffic or data warehousing workload with moderate reporting traffic	Heavy concurrent workload with CPU, memory, and low latency demand

As discussed earlier, the relative transaction performance supported by each service tier is measured in terms of DTUs. While monitoring Azure SQL Database, one of the important metrics to monitor is DTU percentage. The DTU percentage metric gives the DTU for the current workload as a percentage of the maximum DTU supported by that service tier of the database. If the DTU percentage stays consistently at 100, it means it is time to move or upgrade to the next service tier. The following section explains how you can see or monitor the performance metrics for Azure SQL Database.

Monitoring Performance Metrics for Azure SQL Database

In on-premise SQL Server instances as well as SQL Server on Azure VM (IaaS), we use performance monitor data, Dynamic Management Views (DMVs), and Extended Events to monitor the performance metrics for SQL Server instances. Performance monitor (perfmon) counters are primarily useful for monitoring resource utilization on the server and resources consumed by SQL Server. DMVs are useful for tracking wait statistics, user session details, blocking, and more, while extended event sessions are like trace recorders running on the SQL Instance to capture various events or activities as defined in the XEvents session.

Similarly, Azure SQL Database allows you to monitor the database performance using performance monitor data, dynamic management views (DMVs), and Extended Events (introduced in V12). The difference here is you do not need to configure or manage the performance data collections since the data is continuously captured and managed by default. You can view the performance monitor data from Azure Portal by following the steps below.

Monitoring Resource Usage using Performance Charts

1. Log on to Azure Management Portal and click Browse All > SQL databases and select the database you want to monitor. Then, click on Edit on the Monitoring sub-window as shown in the following figure:

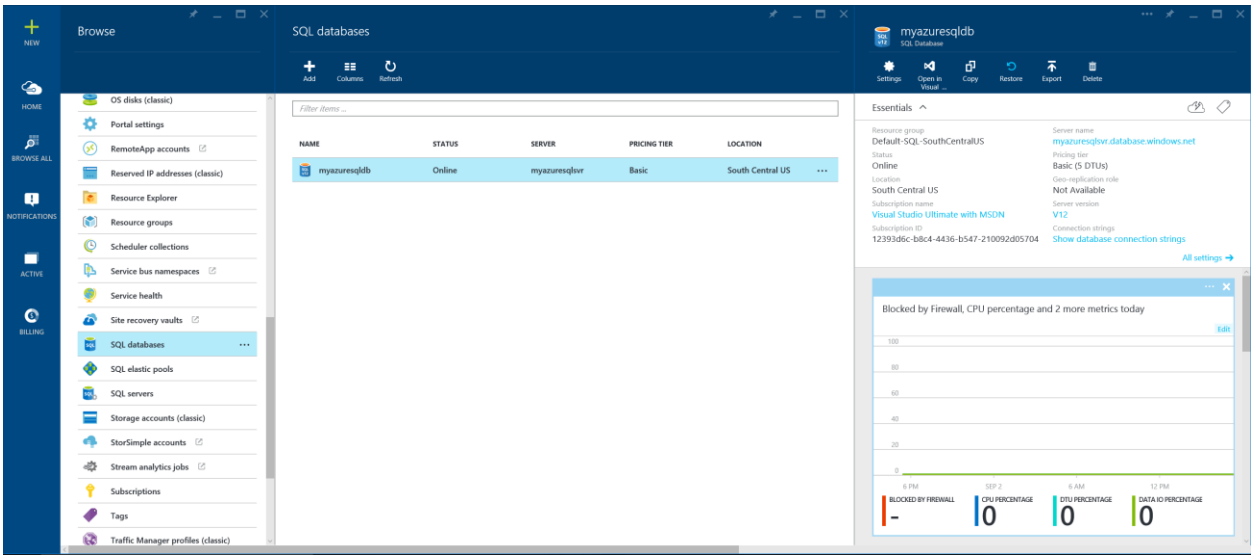

Figure 22: Monitoring Performance Metrics for Azure SQL Database

2. The Edit Chart sub-window allows you to select or deselect various counters, which are collected by default.

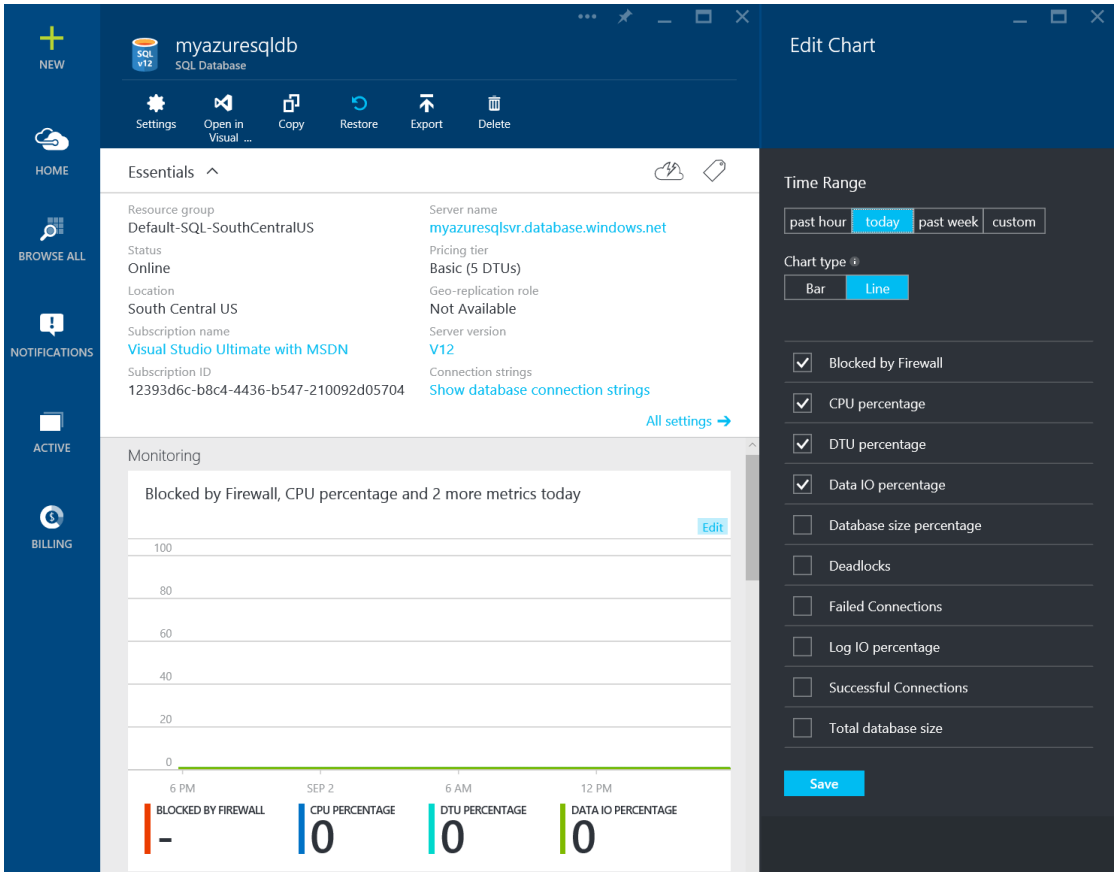

Figure 23: Edit Performance Monitoring Chart

These performance metrics are self-explanatory. As discussed earlier, DTU percentage metrics tell you the DTU of the current database workload as a percentage of max DTU supported by that service tier and is an important metric when considering moving to the next level of service tier. Similarly, database size percentage provides the current size of the database as a percent of the max database size supported by that service tier, another important metric when deciding if the workload should be moved to the next service tier. In general, if you see CPU percentage, DTU percentage, Data IO percentage, Log IO percentage, or database size percentage at 100 consistently, you should consider moving to the next tier so that the throughput of the database is not throttled by tier limitations.

Monitoring Resource Usage using DMVs

You can also access the resource usage data using the **sys.resource_stats** dynamic management view. It provides the same information as the performance monitoring charts.

The following query provides the resource utilization for database myazuresqldb for the past seven days.

Code Listing 26: Monitoring Resource Usage Using DMV

```
SELECT *

FROM sys.resource_stats

WHERE database_name = 'myazuresqldb' AND

      start_time > DATEADD(day, -7, GETDATE())

ORDER BY start_time DESC;
```

In order to evaluate how well your workload fits the performance level, you have to drill down at each different aspect of the resource metrics: CPU, reads, write, number of workers, and number of sessions. Here is a revised query using **sys.resource_stats** to report the average as well as maximum values of these resource metrics.

Code Listing 27: Monitoring Resource Usage Using DMV

```
SELECT

    avg(avg_cpu_percent) AS 'Average CPU Utilization In Percent',

    max(avg_cpu_percent) AS 'Maximum CPU Utilization In Percent',

    avg(avg_physical_data_read_percent) AS 'Average Physical Data Read
Utilization In Percent',

    max(avg_physical_data_read_percent) AS 'Maximum Physical Data Read
Utilization In Percent',

    avg(avg_log_write_percent) AS 'Average Log Write Utilization In
Percent',

    max(avg_log_write_percent) AS 'Maximum Log Write Utilization In
Percent',

    avg(active_session_count) AS 'Average # of Sessions',

    max(active_session_count) AS 'Maximum # of Sessions',

    avg(active_worker_count) AS 'Average # of Workers',

    max(active_worker_count) AS 'Maximum # of Workers'

FROM sys.resource_stats

WHERE database_name = 'myazuresqldb' AND start_time > DATEADD(day, -7,
GETDATE());
```

By monitoring the average CPU and IO trend over a period of time, you can further decide whether it is time to move to the next service tier.

Scaling Up or Down in Azure SQL Database

Before moving between service tiers or performance levels, make sure you have available quota on the server. If you need additional quota, you will have to contact Microsoft customer support to be able to scale up on a server.

You can change the service tier of Azure SQL Database at any time by following the procedure described in this section.

1. Log on to Azure Management Portal and Click Browse all > SQL databases. Select the database whose service tier you would like to change and click on the Pricing Tier on the database sub-window.

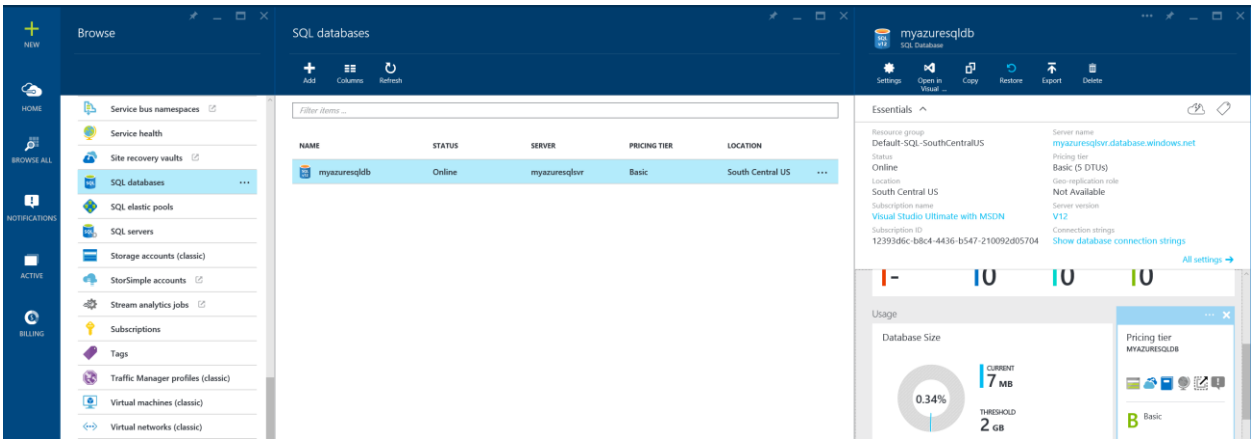

Figure 24: Changing the Service Tier of Azure SQL Database

2. You can select the service tier to which you would like to move and click Select to upgrade to a new service tier.

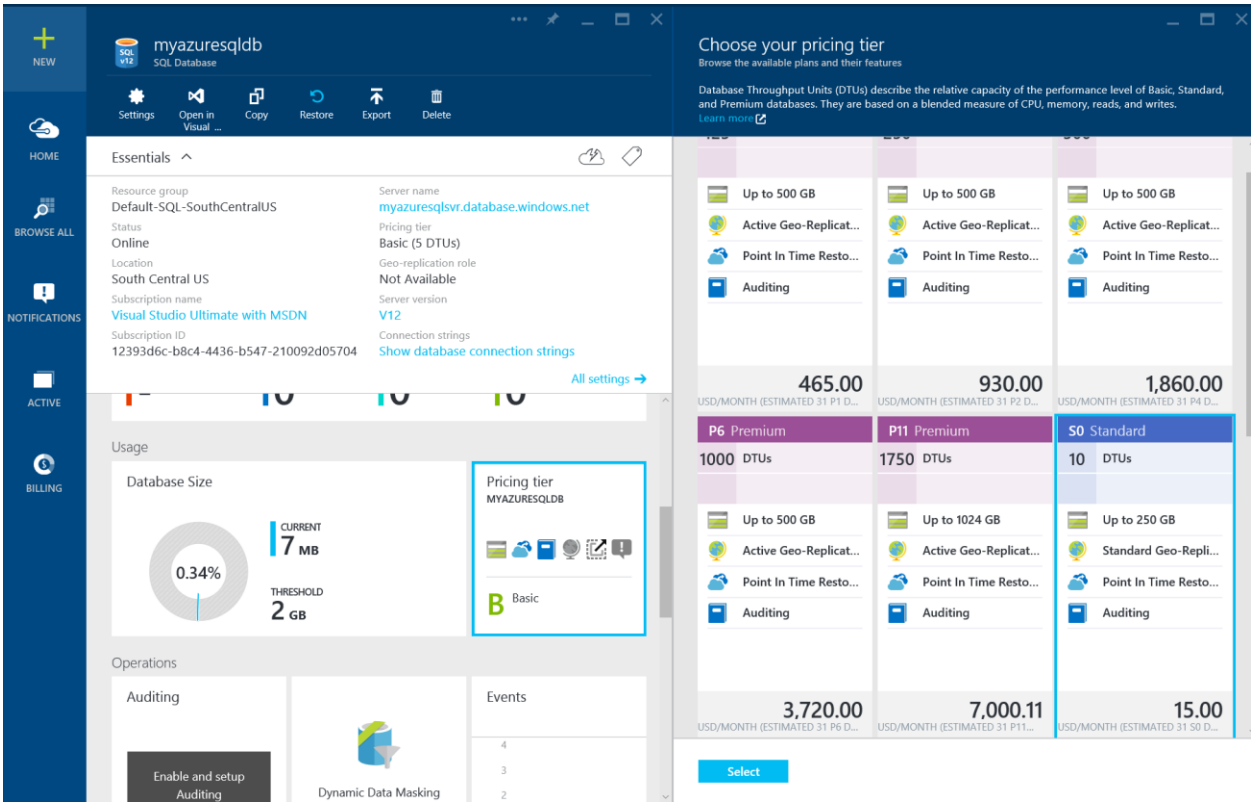

Figure 25: Changing the Service Tier of Azure SQL Database

Changing the service tier of the database is an online operation but should be performed during off-business hours by stopping the application. This will expedite the upgrade operation. The service tier of the database can also be changed using the Set-AzureSqlDatabase PowerShell cmdlet.

Monitoring and Troubleshooting Bad Index Design

A common problem with OLTP database performance is related to bad index design for the database. Often, database schemas are designed and shipped without testing at scale (either in load or in data volume). Unfortunately, the performance of a query plan may be acceptable at small scale but may degrade substantially when faced with production-level data volumes. The most common source of this issue is due to the lack of appropriate indexes to satisfy filters or other restrictions in a query. Often, this manifests as a table scan when an index seek could suffice.

With bad index design for the database, moving to the higher service tier can alleviate the problem a bit. As the database size continues to grow, however, the performance of the queries will continue to degrade, and moving to the next performance tier may not solve the performance issue.

Azure SQL Database contains missing DMV indexes, similar to an on-premise SQL server, to help database administrators find and fix common missing index conditions. This allows a database administrator to quickly guess which index changes might improve the overall workload cost for a given database and its real workload.

The following query can be used to evaluate potential missing indexes.

Code Listing 28: Monitoring Missing Indexes Using DMV

```
SELECT CONVERT (varchar, getdate(), 126) AS runtime,

    mig.index_group_handle, mid.index_handle,

    CONVERT (decimal (28,1), migs.avg_total_user_cost *
migs.avg_user_impact *

            (migs.user_seeks + migs.user_scans)) AS improvement_measure,

    'CREATE INDEX missing_index_' + CONVERT (varchar,
mig.index_group_handle) + '_' +

              CONVERT (varchar, mid.index_handle) + ' ON ' + mid.statement
+ '

              (' + ISNULL (mid.equality_columns,'')

              + CASE WHEN mid.equality_columns IS NOT NULL

                         AND mid.inequality_columns IS NOT NULL

                    THEN ',' ELSE '' END + ISNULL (mid.inequality_columns,
'')

              + ')'

              + ISNULL (' INCLUDE (' + mid.included_columns + ')', '') AS
create_index_statement,

    migs.*,

    mid.database_id,

    mid.[object_id]

FROM sys.dm_db_missing_index_groups AS mig

INNER JOIN sys.dm_db_missing_index_group_stats AS migs

    ON migs.group_handle = mig.index_group_handle

INNER JOIN sys.dm_db_missing_index_details AS mid

    ON mig.index_handle = mid.index_handle
```

```
ORDER BY migs.avg_total_user_cost * migs.avg_user_impact * (migs.user_seeks
+ migs.user_scans) DESC
```

The following query can be used to find the indexes which are updated but never used for seeks, scans, or lookups. The outcome of this query shows indexes that are potential candidates to be dropped since they are overhead for DML operations and not helping at all to expedite the querying of the table.

Code Listing 29: Monitoring Unused Indexes Using DMV

```
SELECT
     DB_NAME()                      AS DatabaseName,
     OBJECT_NAME(i.OBJECT_ID)       AS TableName ,
     i.name                         AS IndexName,
     s.user_updates                 AS IndexUserUpdates

 FROM sys.indexes i
 LEFT JOIN sys.dm_db_index_usage_stats s
     ON s.OBJECT_ID = i.OBJECT_ID
     AND i.index_id = s.index_id
     AND s.database_id = DB_ID()
 WHERE OBJECTPROPERTY(i.OBJECT_ID, 'IsIndexable') = 1
 -- index_usage_stats has no reference to this index (not being used)
 AND s.index_id IS NULL
 -- index is being updated, but not used by seeks/scans/lookups
 OR (
         s.user_updates > 0
         AND s.user_seeks = 0
         AND s.user_scans = 0
         AND s.user_lookups = 0
     )
 ORDER BY OBJECT_NAME(i.OBJECT_ID) ASC
```

Besides the DMV, the Microsoft Product Team has come up with Index Advisor, which is currently in preview at the time of writing. Index Advisor monitors and analyzes the queries hitting the database continuously and recommends indexes which will have maximum positive performance impact. It can automatically create indexes for you and continue to monitor the newly created indexes. If the newly created indexes do not have significant impact, it reverts back and drops the indexes automatically. The feature is still in preview in Azure SQL Database V12, but you can find further details on this feature in the following article:

https://azure.microsoft.com/en-us/documentation/articles/sql-database-index-advisor/

Query Tuning Using Query Store

Another new feature, which is scheduled to be released in the box version of SQL 2016 and is currently in preview in Azure SQL Database V12, is Query Store. As your database goes into production and the real-world data starts populating the database, the queries no longer behave the same as they did in development or testing. When that happens, it takes a long time to identify and troubleshoot the queries which are performing badly due to changes in query plans since you might not have a previous version of the query plan stored in the database. This problem is solved with Query Store. The Query Store feature acts as a figurative airline flight recorder, storing the historical data of the queries and query plans, allowing you to identify and understand the changes in the query behavior and query plan over a period of time.

Query Store is a database-scoped feature and can be turned on at the database level. You can limit the data collected by specifying the max limit for the data capture or sampling frequency. SQL 2016 Management Studio allows you to see the analysis and reporting of query store plans and objects. This allows you to identify the top resource-consuming queries or slow-performing queries over a period of data capture. If your workload contains a query that is being executed with different plans and variable performance, you can use Query Store to force SQL Server to always use the optimal plan in future executions. This is particularly useful for troubleshooting parameter sniffing scenarios in which a stored procedure or a parameterized query uses a different plan for different values of a parameter. If the bad plan gets cached in the SQL Server cachestore first, all the subsequent execution of the stored procedure or query will use the bad plan and perform slowly. Using Query Store, you can force the query to use the optimal plan instead.

Since the feature is still in preview, it is difficult to document the steps to set up and analyze Query Store, but you can find the implementation steps in the following article:

http://azure.microsoft.com/en-us/blog/query-store-a-flight-data-recorder-for-your-database/

Cross-Database Sharding or Scaling Out with Azure SQL Database

If you believe the resource and capacity limits provided by the highest premium service tier in Azure SQL Database may not be sufficient to handle your workload due to high demands from the business or users, you can consider scaling out or a technique commonly known as cross-database sharding. While using the cross-database sharding technique, you can split the data on a single dimension across multiple databases. This is possible because most OLTP applications perform transactions that only apply to a single row or a small group of rows within the schema. For example, if a database contains customer, order, and order details, then this data could be split into multiple databases by grouping a customer with the related order and order-detail information into a single database. The application would split different customers across databases, effectively spreading the load across multiple databases. This allows customers not only to avoid the maximum database size limit, but it also allows Azure SQL Database to process workloads that are significantly larger than the limits of the different performance levels so long as each individual database fits into its DTU.

While database sharding does not reduce the aggregate resource capacity for a solution, this technique is highly effective for supporting very large solutions spread over multiple databases and allows for each database to run at a different performance level to support very large "effective" databases with high resource requirements.

Microsoft recently released the Elastic Database tools for Azure SQL Database, a client library and management tools for creating sharded database solutions on Azure Database. The following article provides more details on getting started with the elastic database tools for Azure SQL Database:

http://azure.microsoft.com/documentation/articles/sql-database-elastic-scale-get-started/

Chapter 9 Security in Azure SQL Database

Security is one of the important considerations when hosting your data in the cloud. Azure SQL Database provides authentication and authorization options to limit access to your database, encryption to protect the data at rest, and auditing capabilities to monitor the data access. In this chapter, you will understand the various security features available in Azure SQL Database and how to leverage them to protect the data in the cloud.

Connection Security

Azure SQL Database restricts the connection to the server and the database by server-level and database-level firewalls. Hence only the IP addresses and address range explicitly specified and whitelisted in the firewall rules are allowed to make a connection to server and database. This limits any unwanted access to the database and, as a best practice, only the application server or client machine IP addresses should be defined in the firewall to allow access to the database. As discussed earlier, you can configure server-level firewall rules and database-level firewall rules using Azure Management Portal, PowerShell, T-SQL, and REST APIs.

Azure SQL Database supports network encryption (SSL/TLS) while data is in transit to and from the database. In your application's connection string, you must specify parameters to encrypt the connection and not to trust the server certificate, otherwise the connection will not verify the identity of the server and will be susceptible to "man-in-the-middle" attacks. When you copy the connection string from Azure Management Portal, it automatically appends the parameter to encrypt the connection and to not trust the server certificate. For the ADO.NET driver, these connection string parameters are Encrypt=True and TrustServerCertificate=False. For more information, see Azure SQL Database Connection Encryption and Certificate Validation.

Authentication

Authentication is the process of validating a user's credentials and allowing entry if the user's credentials are legitimate. Unlike the boxed version of SQL Server, which supports Windows and SQL Authentication, Azure SQL Database currently only supports SQL Authentication. The support for Azure Integrated AD Authentication might come in the future, but currently only SQL Authentication is supported.

When you provision a logical server for hosting your Azure SQL Database, you specify a login and password for the server admin, which acts like a sysadmin on the server and has permissions to create or drop a database and acts as db_owner for all the databases hosted on that server. DBAs or individuals responsible for managing the server can use these credentials to log in to the server to perform admin or management activity on the server.

However, as a best practice, for each application or non-admin connections to the database, you should create a separate account for authentication to limit the permissions granted to the account, thereby minimizing the surface area of attack if that account is compromised for any reason. The preferred approach is to create a contained database user, which can be authenticated by the database itself and which allows the application to directly connect to the database, reducing the risk of malicious activity or SQL injections if the application has vulnerability and if the code is compromised.

You can create a contained database user by executing the following T-SQL while connected to your user database with your server admin login:

Code Listing 30: Creating a Contained Database User

```
CREATE USER user1 WITH password='<Strong_Password>'
```

A strong password is a password that meets the following requirements:

- Does not contain all or part of the user's account name
- Is more than eight characters in length
- Contains characters from at least three of the following categories:
    - English uppercase characters (A through Z)
    - English lowercase characters (a through z)
    - Base 10 digits (0 through 9)
    - Nonalphabetic characters (for example: !, $, #, %)

Additional contained database users can be created by any user with the ALTER ANY USER permission.

Authorization

Authorization is the process of limiting the access of authenticated users to avoid undesired access. After a user is authenticated by providing his or her username and password in the database, the user is authorized by the roles and permissions within the database. As a best practice, you should always grant permission to users with the least required privileges.

Azure SQL Database provides the following options to limit the access of authenticated users.

Database Roles

The database roles in Azure SQLDatabase are no different than database roles available in the on-premise version of SQL Server. Every database user belongs to the public database role. When a user has not been granted or has been denied specific permissions on a securable object, the user inherits the permissions granted to the public on that object. You can add the user to any of the fixed database roles available by default or to flexible database roles which you create. You can create new flexible database roles by grouping the permissions which you would like to grant to the group of users.

You can find further details on database roles in the following MSDN article:

https://msdn.microsoft.com/library/ms189121

Database Permissions

Besides creating and granting database roles to users, you can also grant individual database-level permissions to the users. The database permissions in Azure SQL Database are again similar to the permissions in the on-premise version of SQL Server and are listed in the following MSDN article:

https://msdn.microsoft.com/library/ms191291

If there is a collection of permissions to be assigned to a group of users, it would make sense to create a database role by assigning the permissions to that role and then assigning the group of users to that new role. If there are few users, then it would make sense to assign them granular permissions, thereby giving them the least necessary privilege.

Using Stored Procedure, Impersonation, and Module Signing

Impersonation and module signing are another technique in SQL Server to control the access and securely allow the users to elevate the permissions temporarily in a controlled manner. As a security best practice, it is recommended that applications should access data using a stored procedure. SQL Server ownership chaining allows you to revoke or deny access to the user for the underlying base tables while still granting the user access to the data via the stored procedure. Owner chaining is possible when the stored procedure and the base tables are owned by the same user. However, if they are owned by a different user, you can use EXECUTE AS to impersonate a different user while executing the stored procedure. With this method, the user never has access to the base table and access to the data is controlled by stored procedure execution.

Module signing allows you to sign the stored procedure with a certificate or asymmetric key. This is designed for scenarios in which permissions cannot be inherited through ownership chaining or when the ownership chain is broken, such as dynamic SQL. You then create a user mapped to the certificate, granting the certificate user permissions on the objects the stored procedure needs to access.

When the stored procedure is executed, SQL Server combines the permissions of the certificate user with those of the caller. Unlike the EXECUTE AS clause, it does not change the execution context of the procedure. Built-in functions that return login and user names return the name of the caller, not the certificate user name.

A digital signature is a data digest encrypted with the private key of the signer. The private key ensures that the digital signature is unique to its bearer or owner. You can sign stored procedures, functions, or triggers.

Row-level Security

Row-level security is a new feature that was recently released by Microsoft in Azure SQL Database and is scheduled to be released in the box version of SQL 2016. As the name suggests, it allows you to control the access at the row-level within a table. Previously, the granular level of access control you can grant or deny to the user within a database was table-level. As a result of this, if you did not want a user to see a subset of rows in that table, you would be required to create a separate table or view and grant access to that table or view to the user or control the security from the application layer. Row-level security helps to simplify the physical design of the database by allowing you to store all the data in the same table and access to the rows is evaluated by runtime characteristics of the user. For instance, you can store all employee salary information in a single table and grant the employees row-level access to only their records, allowing them to see only their data. Similarly, you could store all customer data in a single table and grant row-level access to customerid, allowing customers to see only their own records.

You can find further details on the implementation of row-level security in the following MSDN article:

https://msdn.microsoft.com/library/dn765131

Dynamic Data Masking

Azure SQL Database Dynamic Data Masking limits sensitive data exposure by masking it to non-privileged users. Dynamic data masking is currently in preview for all service tiers in the V12 version of Azure SQL Database.

Dynamic data masking helps prevent unauthorized access to sensitive data by enabling developers to designate how much of the sensitive data to reveal with a minimal impact on the application layer. It's a policy-based security feature that hides sensitive data in the result set of a query over designated database fields, while the data in the database is not changed.

For example, a call center support person may identify callers by several digits of their Social Security number or credit card number, but those data items should not be fully exposed to the support person. A developer can define a masking rule to be applied to each query result that masks all but the last four digits of any Social Security number or credit card number in the result set. For another example, by using the appropriate data mask to protect personally identifiable information (PII), a developer can query production environments for troubleshooting purposes without violating compliance regulations.

You can set up dynamic data masking for your database using Azure Management Portal, PowerShell, or REST APIs.

You can use the following steps to configure dynamic data masking:

1. Log on to Azure Portal with admin privileges to your database. Click Browse All >Select SQL Databases and select the database for which you would like to configure Dynamic Data Masking. Click Dynamic Data Masking in the database sub-window as shown in the figure below.

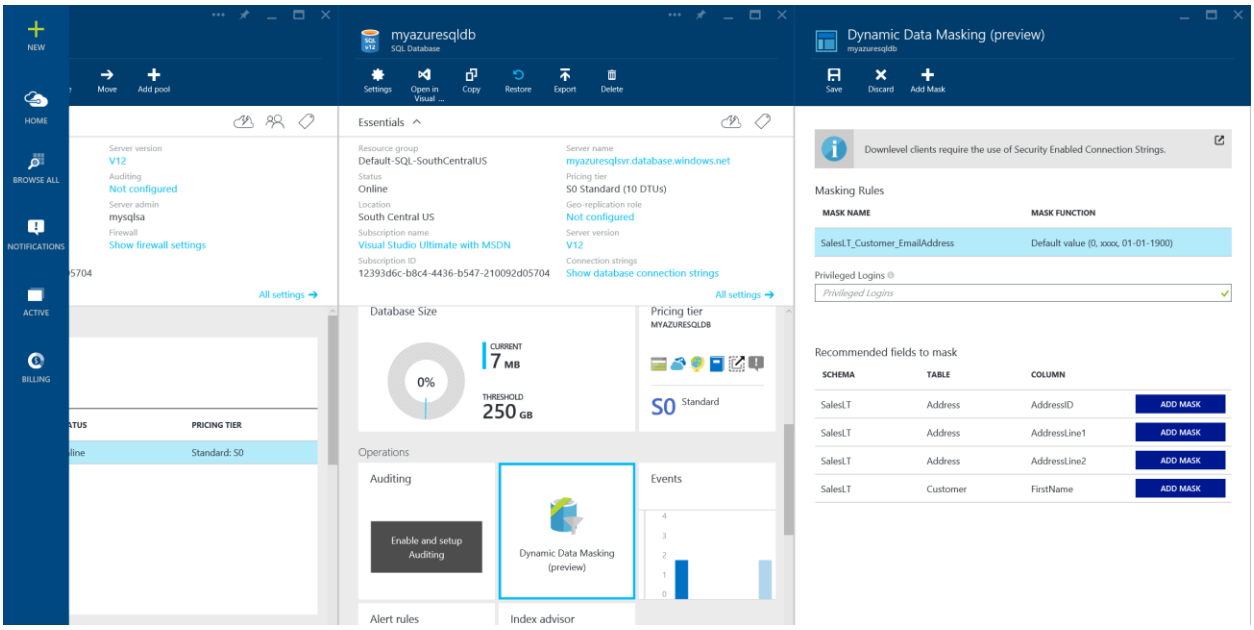

Figure 26: Configuring Dynamic Data Masking

2. In the Dynamic Data Masking configuration sub-window, click Add Mask to open the Add Masking Rule configuration sub-window.

3. Select the Table and Column to define the designated fields that will be masked and choose the masking field format. In the figure below, we choose the Customer table from the SalesLT schema and mask the email address column of the table using the Email Masking format as shown.

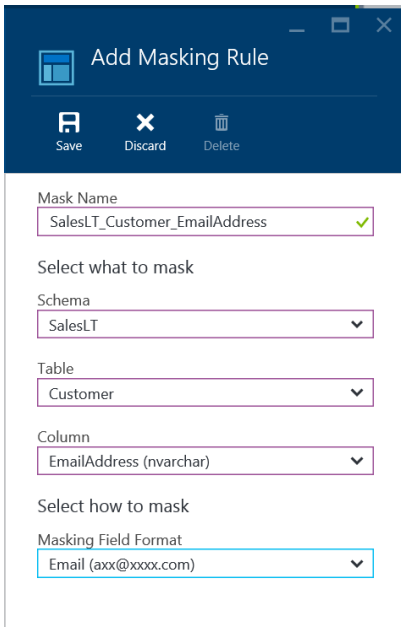

Figure 27: Adding Masking Rule for a Column in the Table

4. Click Save in the data masking rule blade to update the set of masking rules in the dynamic data masking policy and Type the privileged logins that should have access to the unmasked sensitive data.

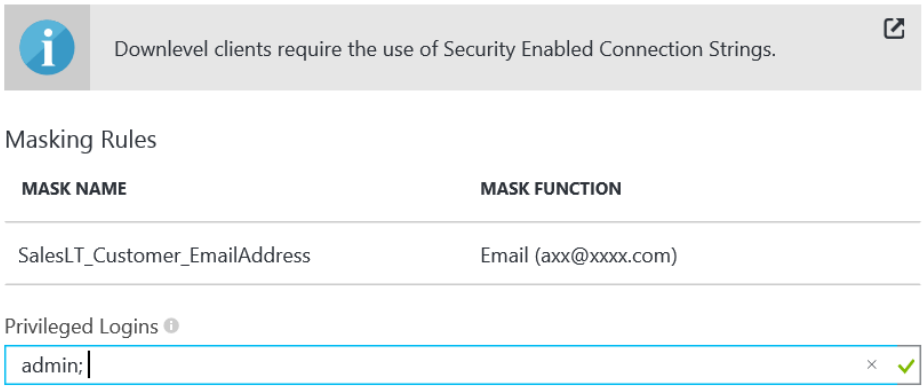

Figure 28: Adding the Privileged Login for Masking Rule

5. Click Save in the data masking configuration sub-window to save the new or updated masking policy.

6. If you are using a Downlevel Client, then you must update existing clients to use a modified connection string format. For more information, see Downlevel clients.

You can also set the data masking policy and rules using the Set-AzureSqlDatabaseMaskingPolicy and Set-AzureSqlDatabaseMaskingRule PowerShell cmdlets.

Encryption

Azure SQL Database supports Transparent Data Encryption, which was first introduced in the box version of SQL 2008 to protect the data when it is at rest or stored in database files and backups. The feature is still in preview and will go live soon in most Azure data center regions. Transparent Data Encryption (TDE), as the name suggests, is transparent to the application and doesn't require any special handing or code changes in the application. TDE encrypts the files of an entire database by using a symmetric key called the database encryption key. The database encryption key is protected by a built-in server certificate. The built-in server certificate is unique for each SQL Database server. If a database is in a Geo Replication relationship, it is protected by a different key on each server. If two databases are connected to the same server, they share the same built-in certificate. Microsoft automatically rotates these certificates at least every 90 days.

In order to enable TDE on Azure SQL Database, you need to log in to the database as admin or as a member of the **dbmanager** role in the master database and execute the following T-SQL code to create a database encryption key and turn on database encryption.

Code Listing 31: Turning on Transparent Data Encryption

```
-- Create the database encryption key that will be used for TDE.

CREATE DATABASE ENCRYPTION KEY

WITH ALGORITHM = AES_256

ENCRYPTION BY SERVER CERTIFICATE ##MS_TdeCertificate##;

-- Enable encryption

ALTER DATABASE [myazuresqldb] SET ENCRYPTION ON;

GO
```

For further details on TDE, you can refer to the following MSDN article:

https://msdn.microsoft.com/library/dn948096.aspx

In addition to TDE, Azure SQL Database also supports column-level encryption to encrypt a column of a table using a symmetric key to prevent any unauthorized access of the data outside the application. This is useful when you do not want DBAs or a high-privileged user to see the data outside of the application.

Auditing

Auditing and tracking database events can help you maintain regulatory compliance and identify suspicious activity. SQL Database Auditing allows you to record events in your database to an audit log in your Azure Storage account. SQL Database Auditing can also be integrated with Microsoft Power BI to facilitate drill-down reports and forensic analytics.

Setting up Auditing on Azure SQL Database

You can follow the steps described in this section to turn ON auditing for your Azure SQL Database.

1. Log on to Azure Portal with admin privileges to your database and click Browse All. Select SQL Databases and select the database for which you would like to configure auditing. Click Settings to open the settings sub-window. Click Auditing to open the auditing sub-window as shown in the figure below.

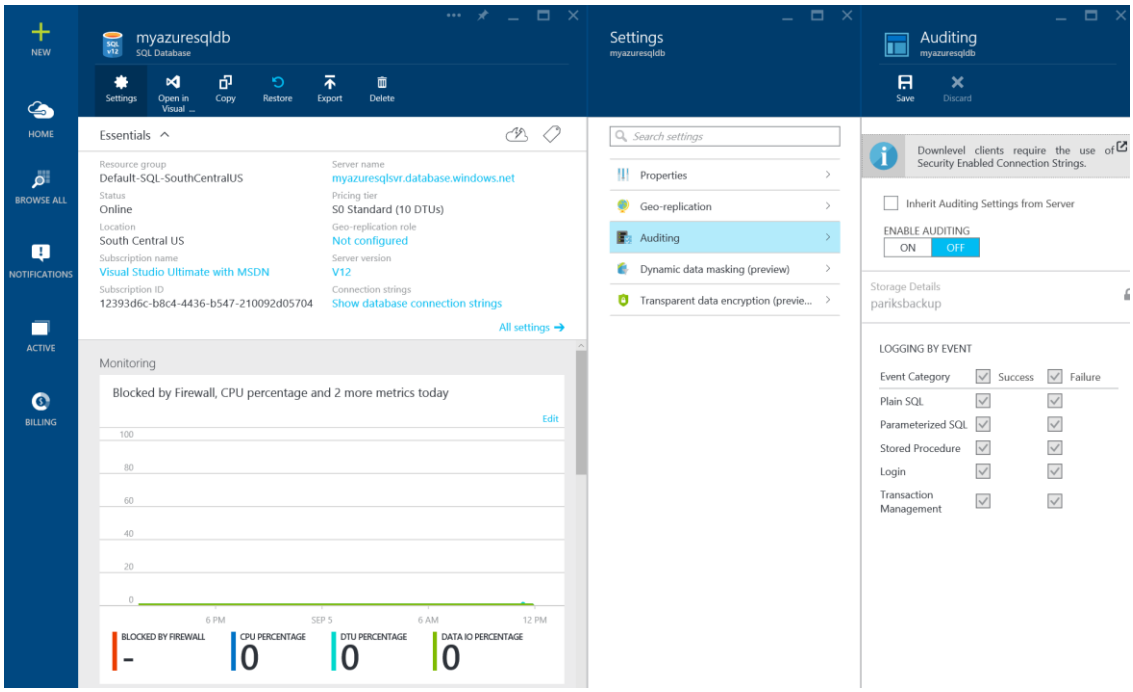

Figure 29: Enabling Auditing on Azure SQL Database

2. In the auditing configuration sub-window, after enabling auditing by clicking ON, select STORAGE DETAILS to open the Audit Logs Storage sub-window. Select the Azure Storage account where logs will be saved and the desired retention period.

Tip: Use the same storage account for all audited databases to get the most out of the preconfigured report templates.

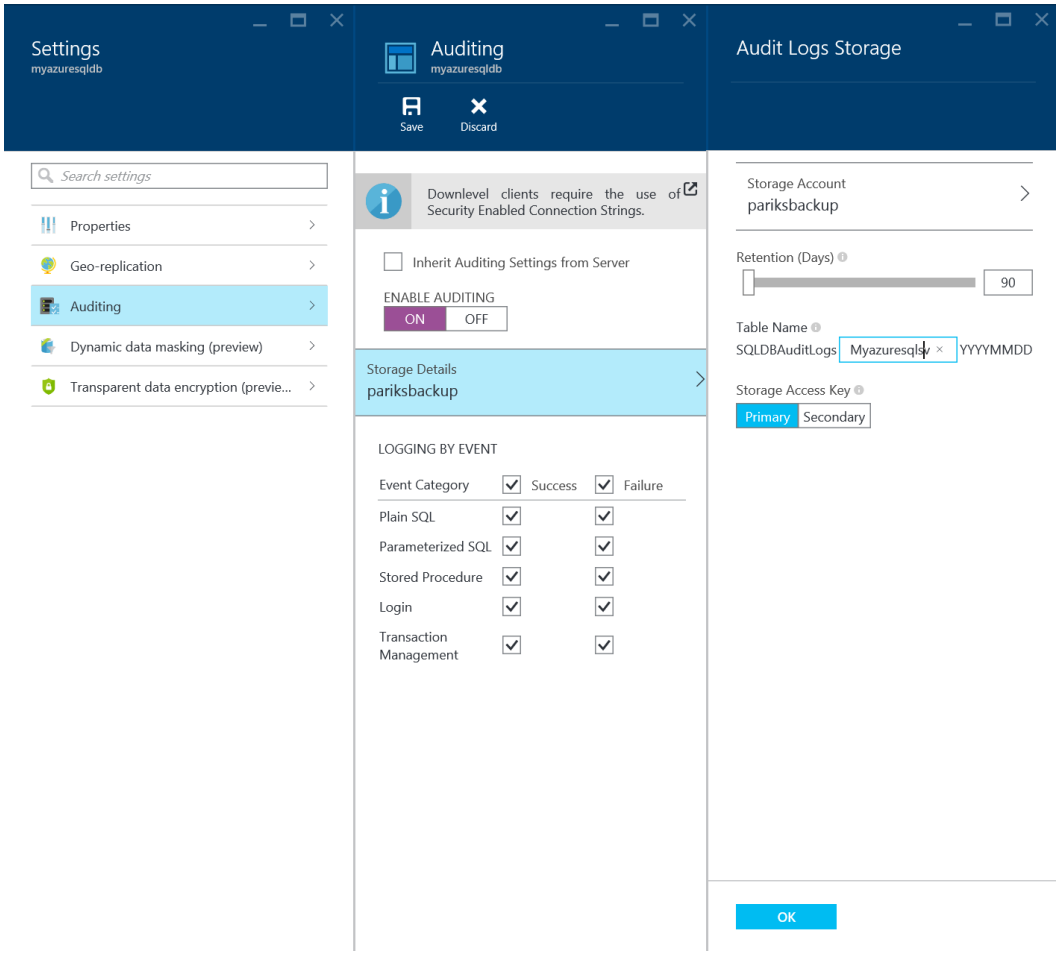

Figure 30: Configuring Auditing Settings for Azure SQL Database

3. Click OK to save the storage settings for Audit Logs Storage.

4. Under LOGGING BY EVENT, click SUCCESS and FAILURE to log all events or choose individual event categories.

5. Click Save to turn ON auditing for Azure SQL Database.

Analyzing the Audit Logs

Audit logs are aggregated in a collection of store tables with a *SQLDBAuditLogs* prefix in the Azure storage account you chose during setup. You can view log files using a tool such as Azure Storage Explorer.

Microsoft Product Team has also created a preconfigured dashboard report template, which is available as a downloadable Excel spreadsheet to help you quickly analyze log data. To use the template on your audit logs, you need Excel 2013 or later and Power Query, which you can download here.

The template has fictional sample data in it, and you can set up Power Query to import your audit log directly from your Azure Storage account.

Chapter 10 Business Continuity with Azure SQL Database

Azure SQL Database has been built from the ground up as a robust, highly available database service using a system in which three or more replicas of each database are maintained at all times, with updates committed to at least two replicas before responding. This HA system addresses local hardware and software failures. In spite of such a high redundancy, it is still susceptible to planned as well as unplanned outages. Microsoft provides an SLA of 99.99% (four nines), which translates to a total downtime of 52.56 minutes per year, but there can still be unplanned outages as the result human error, such as the DBA accidently deleting data while performing a maintenance operation or application upgrade, or as the result of a regional disaster. There can be planned outages as well, such as when you would like to run a database from a different location while the current database is under service maintenance or application upgrade.

As discussed earlier, the following business continuity features are available in V12 version of Azure SQL Database.

Table 12: Business Continuity Features in Azure SQL Database

Capability	Basic tier	Standard tier	Premium tier
Point In Time Restore	Any restore point within 7 days	Any restore point within 14 days	Any restore point within 35 days
Geo-Restore	ERT < 12h, RPO < 1h	ERT < 12h, RPO < 1h	ERT < 12h, RPO < 1h
Standard Geo-Replication	not included	ERT < 30s, RPO < 5s	ERT < 30s, RPO < 5s
Active Geo-Replication	not included	not included	ERT < 30s, RPO < 5s

ERT refers to estimated recovery time, which is the estimated time to restore and recover the database until a recovery point.

RPO stands for Recovery Point Objective, which measures the amount of data lost by the time the database is restored and brought online.

The combination of geo-restore, standard geo-replication, and active geo-replication provides you with a range of options to implement a business continuity solution that meets the needs of your application and business. The following table summarizes some common business continuity scenarios and which solution would be useful under each scenario.

Table 13: Business Continuity Scenarios

Scenario	Geo-restore	Standard Geo-replication	Active Geo-replication
Regional disaster	Yes	Yes	Yes
DR drill	Yes	Yes	Yes
Online application upgrade	No	No	Yes
Online application relocation	No	No	Yes
Read load balancing	No	No	Yes

In this chapter, we will discuss all the business continuity solutions available in Azure SQL Database in detail.

Point-in-Time Restore

A database in Azure SQL Database is continuously backed up within Azure data centers using full database backups as well as incremental and transaction log backup. The point-in-time restore allows you to restore the database from the backups to when the transaction occurred at the specified point in time. This capability is available in all service tiers, but the restore point depends on the retention policy supported by each service tier. You can go back up to seven days with the Basic tier, 14 days with the Standard tier, and 35 days with the Premium tier. Point-in-time restorations are specifically useful for recovering from human error, such as when a user has accidently dropped a table or overwritten data which needs to be recovered. You can perform point-in-time restore using Azure Portal or PowerShell.

You can follow the steps mentioned in this section to restore a database to a specified point in time.

1. Log on to Azure Portal. Click Browse All and select SQL database. Select the database which you would like to restore, and at the top of the database sub-window click Restore as shown in the following figure.

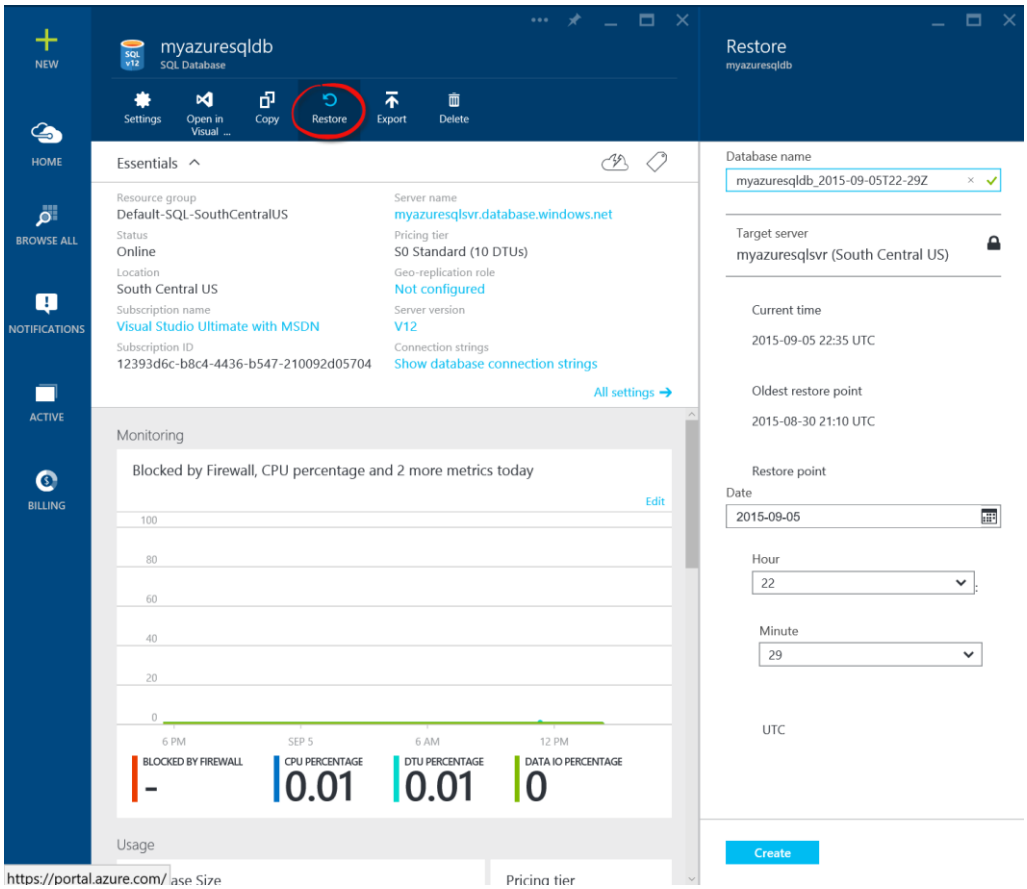

Figure 31: Point in Time Restore of Azure SQL Database

2. Specify the database name and point in time and click Create to restore the database with a new name to the specified point in time.

3. The restore process can be monitored from the notifications tab on the left.

4. After the database is restored, you can see and connect to the database like any other Azure SQL database from SSMS and recover the desired data.

You can also restore the database to a specified point in time using the Start-AzureSqlDatabaseRestore PowerShell cmdlet.

Geo-Restore

Geo-restore provides the ability to restore a database from a geo-redundant backup to create a new database. The database can be created on any server in any Azure region. Because it uses a geo-redundant backup as its source, it can be used to recover a database even if the database is inaccessible due to an outage. Geo-restore is automatically enabled for all service tiers at no extra cost.

You can follow the steps mentioned in this section to perform a geo-restore of an Azure SQL database.

1. Log in to Azure Portal and click New at the top left corner of the portal.

2. Select Data and Storage and then select SQL Database.

3. Provide the database name and the logical server name you would like to restore and select Backup as the source. In the backup sub-window, select the geo-redundant backup which you would like to restore as shown in the following figure.

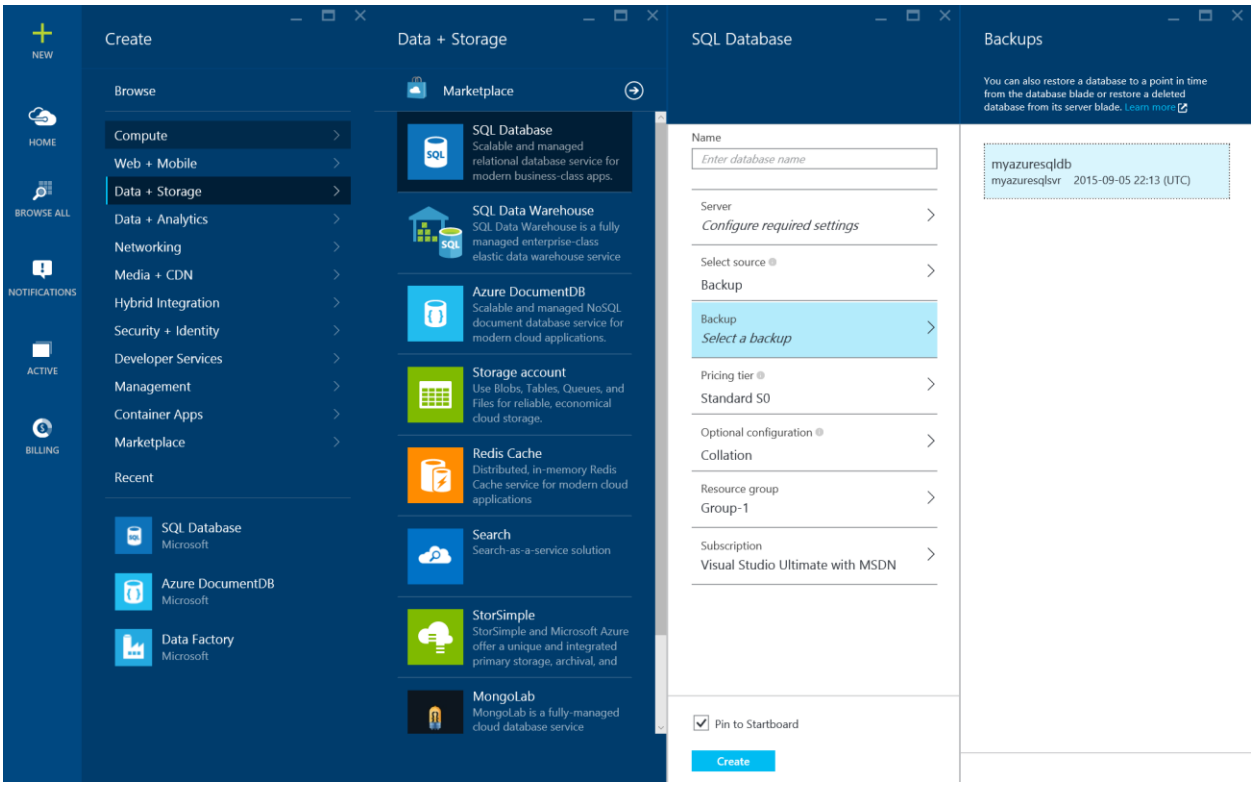

Figure 32: Performing a Geo-Restore of Azure SQL Database

4. Specify the rest of the database properties and click Create.

5. The database restore process will begin and can be monitored using the NOTIFICATIONS tab on the left side of the screen.

6. After the database is restored, you can connect to the database using SSMS by configuring the server-level and database-level firewall rules for your client IP.

You can also perform geo-restore of Azure SQL Database using the Start-AzureSqlDatabaseRecovery PowerShell cmdlet.

Standard Geo-Replication

Standard geo-replication allows you to configure your database to asynchronously replicate committed transactions from the primary database to the secondary in a predefined Azure region. Standard geo-replication is built on the same technology as active geo-replication, but it is optimized for applications that use geo-replication only to protect the application from regional failures. The following list shows how standard geo-replication is different from active geo-replication:

1. Only one secondary database can be created in a Microsoft-defined “DR paired” Azure region.

2. The secondary database is visible in the master database but cannot be directly connected to until failover is completed (offline secondary).

3. The secondary database is charged at a discounted rate as it is not readable (offline).

Standard geo-replication is useful when you want to recover from a regional disaster or during DR drills for compliance.

You can follow the steps in this section to configure standard or active geo-replication.

1. Log in to Azure Portal and click Browse All. Select SQL Database and select the database for which you would like to configure geo-replication.

2. In the database sub-window, scroll down to the bottom until you see Configure Geo-replication and click on it.

3. In the geo-replication sub-window, you will see the Azure regions where you could host your secondary database along with a recommended region. A recommended region is a preferred region for replication in terms of performance, but you can choose the region which you have in mind. If you do not have any preference for Azure location for the secondary database, you can choose the recommended Azure region.

4. Choosing the region will open up a Create Secondary sub-window where you can specify the secondary type as readable or non-readable. Choosing non-readable makes it a standard geo-replication, while readable makes it an active geo-replication.

5. Finally, you can configure the server settings where the secondary database will be hosted. Click Create to configure the secondary database for geo-replication.

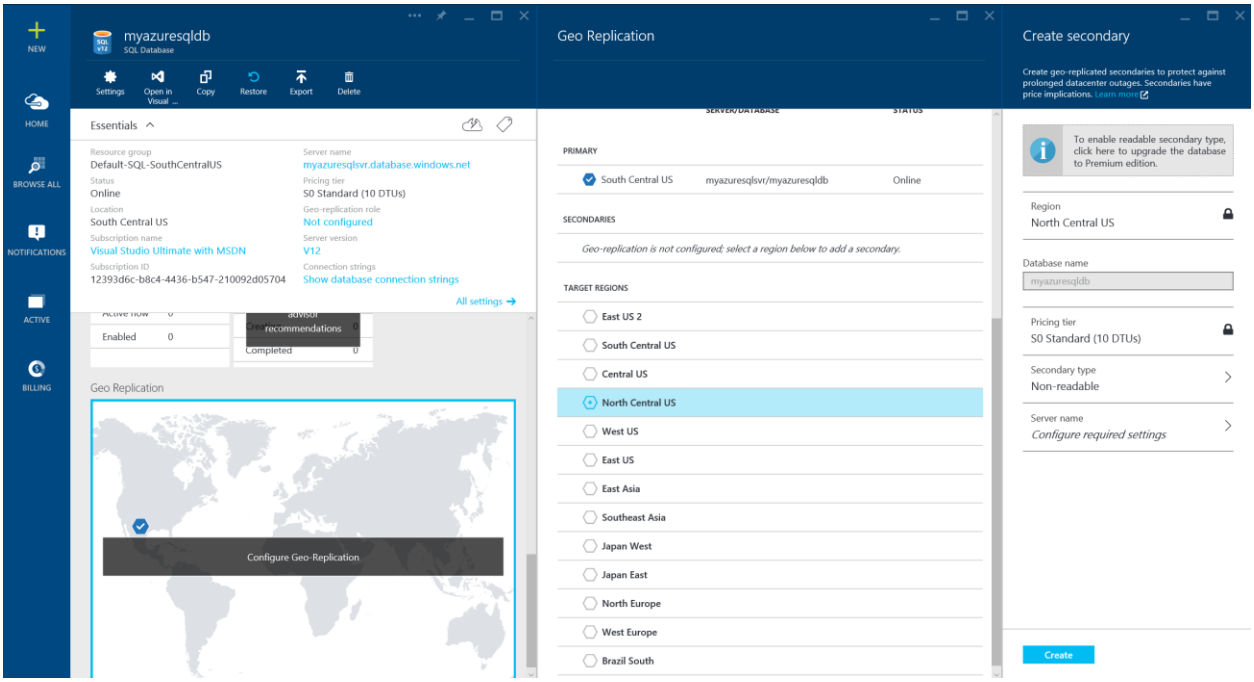

Figure 33: Configuring Standard or Active Geo-Replication

6. Once the secondary replica is created, you will see it listed under the Secondaries section in the geo-replication sub-window.

In the event of outage on the primary database when you want to perform failover, you would come to the same geo-replication sub-window as shown in the previous figure and, under the Secondaries section, right-click on the row with the name of the database you want to recover and click **Stop** to terminate the continuous copy relationship and bring the secondary database online. For a full description on termination of continuous copy relationships, you can refer to the following MSDN article:

https://azure.microsoft.com/en-us/documentation/articles/sql-database-disaster-recovery/

You can also terminate the continuous copy relationship to stop and failover to secondary using the Stop-AzureSqlDatabaseCopy PowerShell cmdlet.

Active Geo-Replication

Active geo-replication, available for Premium databases, provides the richest solution with the least risk of data loss and the most rapid recovery time. It extends standard geo-replication with up to four geo-replicated secondaries that are online and readable at all times, which can also be used for load balancing or to provide low-latency access to replicated data anywhere in the world.

Using active geo-replication, you can create a continuously replicated copy of a database that can be frozen immediately prior to applying updates or maintenance to the database or an application. Should any errors be detected during or after the process, it is easy to fall back quickly to this copy.

> ***Tip: While active geo-replication is available in the Premium service tier, you can still leverage the benefits of active geo-replication for performing application or service upgrades by switching to the Premium service tier just before the upgrade operation, perform the upgrade with active geo-replication, and then switch back to a lower service tier to save cost.***

You can configure active geo-replication using the steps mentioned above when you configure standard geo-replication, but with active geo-replication you will configure the secondary type as readable and can have up to four geo-replicated secondaries that are online and readable at all times. Similar to standard geo-replication, when you would like to failover you would go to the geo-replication sub-window for that database and right-click on the secondary to which you want to failover. Click Stop to terminate the continuous copy relationship and bring the database online in read-write mode.

Once you have managed the failover process, you will want to reconstruct the same pattern of geo-replication relationships you were using previously, only with the new production database as the primary to ensure you have the geo-redundancy and load-balancing that your applications and business continuity policies require.

For further information on Active replication for Azure SQL Database, you can refer to the following MSDN article:

https://azure.microsoft.com/en-us/documentation/articles/sql-database-geo-replication-overview/