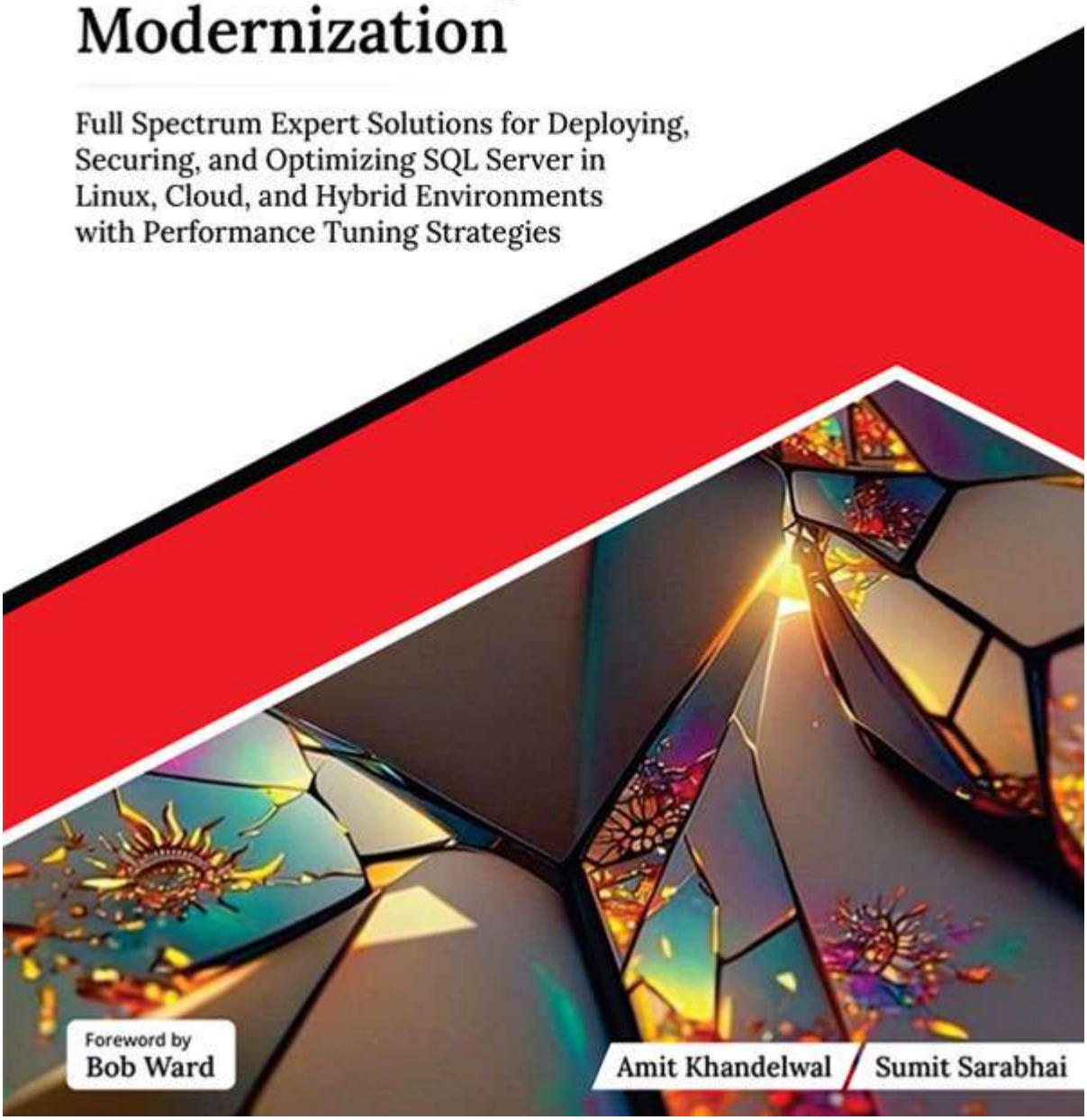
ULTIMATE

SQL Server and Azure SQL for Data Management and Modernization

Full Spectrum Expert Solutions for Deploying, Securing, and Optimizing SQL Server in Linux, Cloud, and Hybrid Environments with Performance Tuning Strategies



Foreword by Bob Ward

Amit Khandelwal / Sumit Sarabhai

Ultimate SQL Server and Azure SQL for Data Management and Modernization

---

Full Spectrum Expert Solutions for Deploying, Securing, and Optimizing SQL Server in Linux, Cloud, and Hybrid Environments with Performance Tuning Strategies

---

Amit Khandelwal

Sumit Sarabhai

www.orangeava.com

First Published: October 2024

Published by: Orange Education Pvt Ltd, AVA™

Address: 9, Daryaganj, Delhi, 110002, India

275 New North Road Islington Suite 1314 London,

N1 7AA, United Kingdom

ISBN (PBK): 978-93-48107-84-8

ISBN (E-BOOK): 978-93-48107-23-7

Scan the QR code to explore our entire catalogue

www.orangeava.com

Dedicated To

My Beloved Parents, for Their Unwavering Guidance and Support Throughout my Journey

Shri Suresh Chand Khandelwal

Smt. Shashi Khandelwal

and

My beautiful Wife Swati Whose Love and Partnership have been my Greatest Strength

My Lovely Daughters Shivi and Praavi, You are my Pride and Joy

- Amit Khandelwal

My Beloved Mother, Smt. Sadhana Sarabhai, whose Unwavering Love and Guidance have been my Constant Source of Strength

My lovely wife, Kritika, for her Endless Support and Companionship

And to my Children, Krivi and Krishiv, who Inspire me Every Day with their Joy and Curiosity

- Sumit Sarabhai

FOREWORD

-by Bob Ward

(Principal Architect, Microsoft)

Having worked at Microsoft for over 30 years, focusing on SQL capabilities ground to cloud, I always relish the chance to read a book that covers the "core" of SQL Server-whether it is on Windows Server, Linux systems, a Kubernetes cluster, or on a public cloud like Azure. This book, authored by two SQL experts at Microsoft, Amit and Sumit, does just that. The words "data management and modernization" in the title are especially appealing as you will find this theme throughout the book.

I really like how this book is structured.It starts by teaching the fundamentals and core concepts of the SQL Server engine, including key internal concepts and a nice tour of SQL Server 2022. With this knowledge, the book then guides to understand various deployment options from ground to cloud, including how to use Azure Arc to connect to the cloud. Amit and Sumit have a deep knowledge of Linux, containers, and Kubernetes, giving you an opportunity to learn how to deploy SQL Server in these modern platform environments.

As I read through the core chapters of the book, I get a very deep understanding of security, high availability, and disaster recovery for SQL Server across different deployment options I have already learned. I really like that I have been able to build up my knowledge with plenty of examples and learn how to ensure my data is secure and available for any business requirements I may have.

One of my requirements could be that I need to migrate to the cloud or upgrade to a new version of SQL Server.this book provides guidance on all the options available including how to plan out a migration and the tools needed to make the entire process successful end-to-end. Armed with all this knowledge, the book does an excellent job of providing me best practices and modernization techniques for configuration, security, performance tuning, and monitoring.

Throughout this book I felt I could learn in a highly organized manner but dive deeper with visualizations and examples that clarify the details. You can tell the authors have poured their expertise from working with the Microsoft product team into every chapter of this book. I always love to see a passion for a product I have used throughout my career and this book does all of that and more. I hope you enjoy reading it as much as I did.

About the Authors

Amit Khandelwal is a Principal Product Manager at Microsoft with over 15 years of experience. He has been pivotal in the development and evolution of SQL Server on Linux, significantly contributing to Microsoft's cross-platform solutions. Starting as an intern, Amit has held roles including Support Engineer, Technical Lead, Support Escalation Engineer, and Premier Field Engineer, providing critical support and resolving complex technical issues for enterprise customers with SQL Server implementations and resolutions.

Amit, a tech leader specializing in AI and SQL Server, has expanded SQL Server's reach across various platforms. His expertise drives product strategy and successful deployments. A frequent speaker at industry conferences, Amit shares insights on technology trends. Notable presentations include his sessions at DevDays Asia (Taipei), SQL BITS (London), PASS Data Community Summit (Seattle), Red Hat Summit, SUSECON, Internal Microsoft Events, Bangalore UG group, Data Exposed, and DataMinds. His work inspires the future of tech at Microsoft and beyond.

Sumit Sarabhai is a technology expert with over 17 years of experience at Microsoft, specializing in SQL Server and its evolving technologies. His expertise spans a range of fields, including SQL Product Engineering, Consulting, Pre-Sales, Services, and Technical Support. Sumit possesses a strong command of cloud solution architecture, implementation, capacity planning, risk assessment, and migration strategies.

Over the years, Sumit has played a pivotal role in guiding customers modernize workloads, capacity planning, and successful migrations to the Azure Cloud and Data platforms. His expertise in risk assessment and migration strategies ensures smooth transitions and optimized cloud solutions, earning him recognition for his deep technical knowledge and problem-solving skills.

Currently, Sumit holds the position of Senior Software Engineer, leading the development and maintenance of the Python driver for SQL Server. His expertise ensures efficient database communication between SQL Server and Python applications, supporting developers in building data-driven solutions.

About the Technical Reviewer

Ramesh a highly motivated software engineer with over 14 years of IT experience, recently published his first book, Ultimate Node.js for Cross-Platform App Development with Orange Education Pvt Ltd.

His educational background boasts an M.Tech in Computer Science from BITS-Pilani and a B.Tech from GNIT. Currently, Ramesh serves as an Engineering Manager at Moback Technologies. His expertise lies in full-stack development, having worked extensively with product-based companies on projects utilizing Asp.net, .net core, NodeJS, and popular JavaScript frameworks such as Angular and React.

Kumar's leadership extends beyond management. He actively contributes to development by working on features, conducting code reviews, and resolving technical roadblocks. His skillset encompasses both back-end technologies such as C#, Asp.net Core, GIT, and SQL Server, and front-end technologies such as JQuery, Angular, and React.

A keen adopter of new technologies, he has hands-on experience with DevOps tools such as TFS, Git, and Azure DevOps. Rounding out his impressive qualifications, Kumar possesses extensive experience working on Agile-based projects, consistently delivering high-quality products on time. Additionally, he has explored various cloud platforms such as Azure and AWS.

Acknowledgements

My heartfelt thanks goes to my entire family—my parents and my wife, who shouldered household responsibilities while I balanced work and writing.

I am equally indebted to my brother, Vineet Khandelwal, and his family for caring for the kids during those intense writing sessions. My niece, Krishvi, with her infectious smile and boundless cheerfulness, also provided invaluable support.

Additionally, I want to express my appreciation to Microsoft for supporting my professional development and contributions to SQL Server over the past 15 years. Special thanks to my manager, Tejas Shah, as well as my colleagues, friends, and partners—such as OJ Ngo from DH2i—who diligently reviewed the chapters. Their collective support was instrumental in bringing this book to fruition.

Lastly, I am immensely grateful to the AVA team. Their collaborative efforts transformed disparate chapters into a cohesive book. Without them, this endeavor would have remained an insurmountable challenge.

- Amit Khandelwal

First and foremost, I would like to extend my deepest gratitude to my family. To my mother, for instilling in me the values of perseverance and hard work that have shaped who I am today. To my wife, thank you for your unwavering support, understanding, and patience, especially during

the long hours I dedicated to this book. My siblings, Shivani and Rohit, offered invaluable guidance and inspiration.

I am also deeply grateful to my friends, colleagues, and mentors for their valuable feedback and thoughtful suggestions. Their wisdom and support have been instrumental in my personal and professional growth.

Lastly, I am deeply grateful to the AVA team. Their collaborative spirit and dedication transformed individual chapters into a cohesive, polished book. Without their tireless efforts, this endeavor would have been an overwhelming challenge. Thank you for making this possible.

Thank you all for being part of this journey.

- Sumit Sarabhai

Preface

In the pages that follow, we embark on a remarkable journey—a culmination of our collective experiences. This book is our attempt to share the captivating story of how a product evolved, feature by feature, across diverse deployments.

Our approach in this book is practical. We start with the basics: understanding user objects, system tables, indexes, data pages, and log files in SQL Server. From there, we delve into the rich variety of SQL offerings available today—whether on-premises, in the cloud, or within containers. We explore deployment techniques across platforms, helping you choose the right SQL environment for your specific needs. But that's not all. We cover critical topics: securing SQL Server, configuring high availability and disaster recovery, modernizing databases, optimizing performance, and mastering monitoring and troubleshooting.

So, dear reader, join us on this voyage. Let us unravel the layers of SQL Server, celebrate its growth, and learn together.

This book is basically divided into 8 chapters, listed as follows:

Chapter 1. SQL Server – The In this chapter, we lay the groundwork for understanding SQL Server and its core database engine, which underpins all SQL products, including Azure SQL Database and SQL Server on Linux. This chapter covers essential topics such as SQL Server internals, databases and tables, data and log files, indexes, and backup and restore

processes. Mastering these fundamentals will prepare you for exploring the wider SQL ecosystem, including SQL Server 2022.

Chapters 2. and 3. Realms of SQL: The Realms of SQL explores SQL Server's extensive ecosystem, which has evolved over two decades to meet the diverse needs of users, especially with the rise of cloud computing. This chapter is divided into two parts, covering the SQL ecosystem, types of workloads, considerations for data size and scalability, and choices between modern cloud-based and traditional applications, including hybrid deployments with Azure Arc. By understanding these elements, you can make informed decisions for your SQL Server strategy.

Chapter 4. SQL Server and Security Hand in Glove: In today's digital world, security is vital to database management. This chapter covers SQL Server's key security features, including encryption, authentication, and access control. It demonstrates how these measures work together to protect sensitive data and ensure compliance with modern security standards, offering practical steps to secure your SQL Server environment from both internal and external threats.

Chapter 5. Business Continuity Strategies for SQL Offerings: Ensuring continuous availability of SQL Server databases is vital for business resilience. This chapter covers key approaches such as backup strategies, high availability, disaster recovery, and failover solutions. By the end, you'll be equipped with practical techniques to maintain uptime and protect your data against unforeseen disruptions.

Chapter 6. Accelerate SQL Server Modernization: In an era of rapid technological advancement, modernizing your SQL Server infrastructure is essential for improving performance, scalability, and security. This

chapter focuses on strategies and best practices for upgrading to modern versions of SQL Server, leveraging cloud integration, and optimizing performance. By the end, you'll understand how to streamline your modernization journey for a future-ready SQL Server environment.

Chapter 7. Achieving SQL Brilliance Through Best Practices: Mastering SQL Server requires more than just technical know-how; it demands adherence to proven best practices. In this chapter, we explore essential guidelines for designing, maintaining, and optimizing your SQL Server environment. By following these best practices, you can ensure consistency, performance, and long-term success in managing your databases.

Chapter 8. Monitoring, Performance Tuning and Optimization: Efficient SQL Server performance is critical for business success. This chapter delves into tools and techniques for monitoring system health, fine-tuning performance, and optimizing queries. By focusing on practical strategies, this chapter equips you with the knowledge to maintain peak database performance and swiftly resolve bottlenecks.

Downloading the code

bundles and colored images

Please follow the link or scan the QR code to download the Code Bundles and Images of the book:

https://github.com/ava-orange-education/Ultimate-SQL-Server-and-Azure-SQL-for-Data-Management-and-Modernization

The code bundles and images of the book are also hosted on https://rebrand.ly/5bea4rt

In case there’s an update to the code, it will be updated on the existing GitHub repository.

Errata

We take immense pride in our work at Orange Education Pvt Ltd and follow best practices to ensure the accuracy of our content to provide an indulging reading experience to our subscribers. Our readers are our mirrors, and we use their inputs to reflect and improve upon human errors, if any, that may have occurred during the publishing processes involved. To let us maintain the quality and help us reach out to any readers who might be having difficulties due to any unforeseen errors, please write to us at :

errata@orangeava.com

Your support, suggestions, and feedback are highly appreciated.

DID YOU KNOW

Did you know that Orange Education Pvt Ltd offers eBook versions of every book published, with PDF and ePub files available? You can upgrade to the eBook version at www.orangeava.com and as a print book customer, you are entitled to a discount on the eBook copy. Get in touch with us at: info@orangeava.com for more details.

At you can also read a collection of free technical articles, sign up for a range of free newsletters, and receive exclusive discounts and offers on AVA™ Books and eBooks.

PIRACY

If you come across any illegal copies of our works in any form on the internet, we would be grateful if you would provide us with the location address or website name. Please contact us at info@orangeava.com with a link to the material.

ARE YOU INTERESTED IN AUTHORING WITH US?

If there is a topic that you have expertise in, and you are interested in either writing or contributing to a book, please write to us at We are on a journey to help developers and tech professionals to gain insights on the present technological advancements and innovations happening across the globe and build a community that believes Knowledge is best acquired by sharing and learning with others. Please reach out to us to learn what our

audience demands and how you can be part of this educational reform. We also welcome ideas from tech experts and help them build learning and development content for their domains.

REVIEWS

Please leave a review. Once you have read and used this book, why not leave a review on the site that you purchased it from? Potential readers can then see and use your unbiased opinion to make purchase decisions. We at Orange Education would love to know what you think about our products, and our authors can learn from your feedback. Thank you!

For more information about Orange Education, please visit

Table of Contents

CHAPTER 1

SQL Server – The Fundamentals

Introduction

This introductory chapter helps you to understand the basics of SQL Server, as knowing the core SQL Server database engine gives you a grasp on the fundamentals. The same SQL Server engine runs all the SQL products, such as Azure SQL Database, Azure SQL Managed Instance, Azure SQL Server Virtual Machines, Azure SQL edge, and SQL Server on Linux as shown in Figure Hence, understanding the fundamentals of core SQL Server database engine assists you in your journey to learn other SQL suite of products as well.

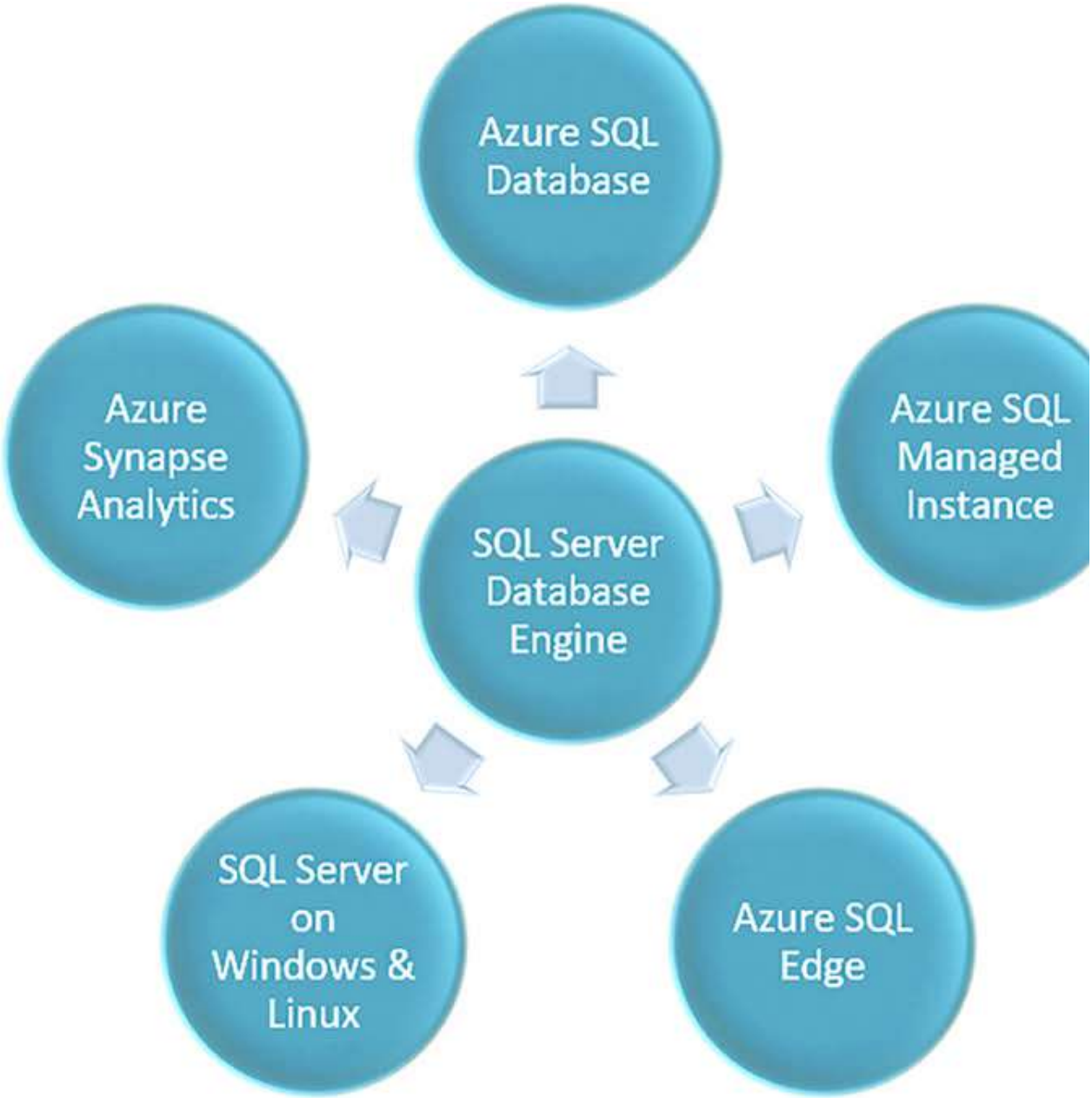


Figure 1.1: SQL Server Database engine runs all the SQL products

Structure

In this chapter, the following topics will be covered:

Understanding SQL Server

SQL server Internals

Databases, tables

Database data and log files

Indexes

Backup and Restore

Introduction to SQL Server 2022

Understanding SQL Server

Microsoft SQL Server is a relation database management system (RDBMS) and like every RDBMS database, it stores data in a tabular format. Each SQL Server process has an instance that you can find in the task manager. You can have more than one SQL Server instance on a Windows machine, and each one has its own process. You can choose between a named instance or a default instance when you install SQL Server. The default instance has the same name as the host machine, while the named instance has a name that you specify, which is shown as For example, in Figure we have installed two named instances of SQL Server: mypc\SQLINSTANCE1 and Where ‘mypc’ is the hostname on which these two instances are installed.

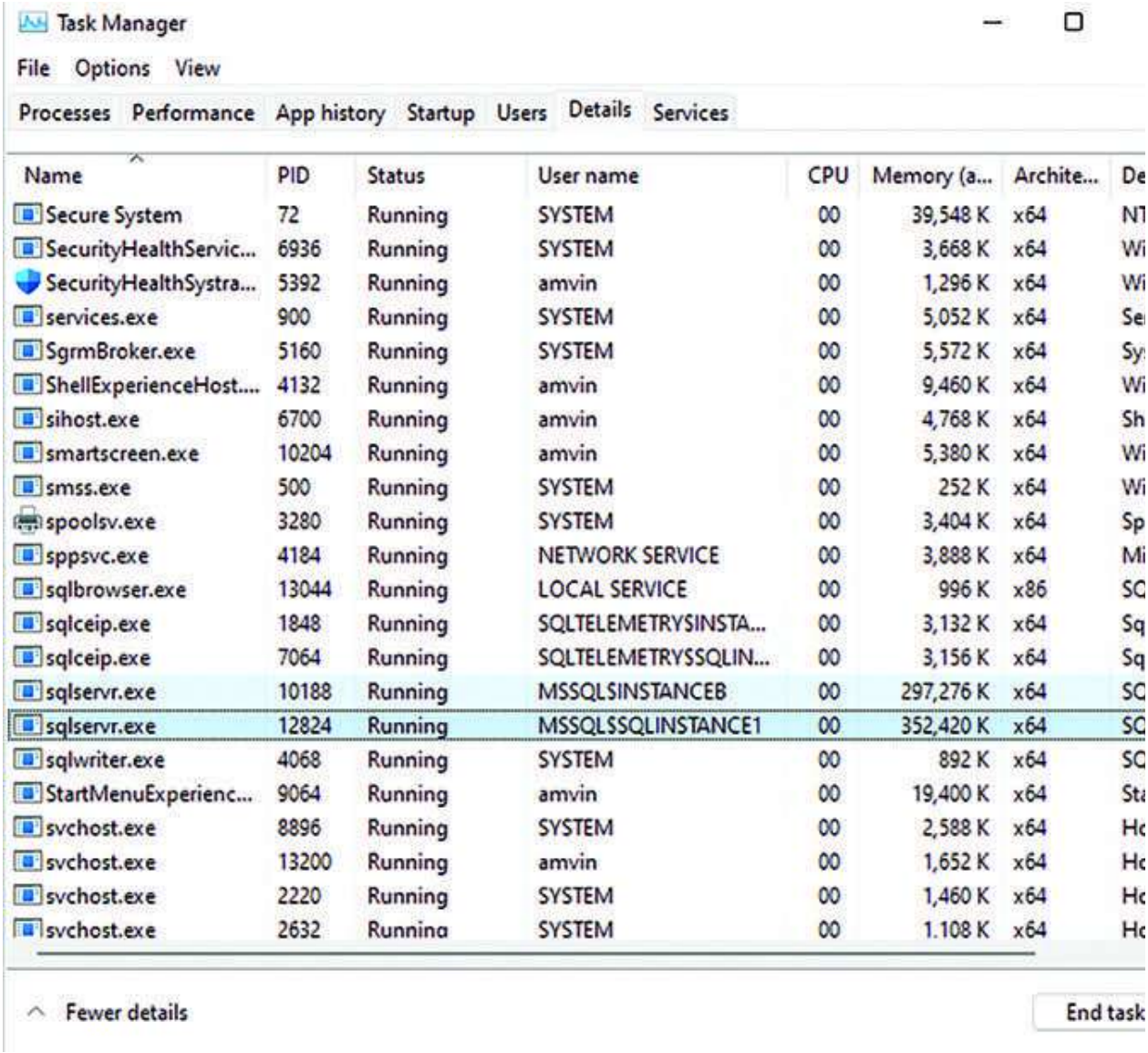


Figure 1.2: SQL Server process listing in Windows 1

On Windows-based machine, you can have more than one instance of SQL Server, while on Linux deployments you can install only one instance of SQL Server which is a default instance. To install more than one instance of SQL Server on Linux, you must use containers. We will learn more about SQL Server on Linux in later chapters.

You can create many databases, Logins, Credentials, Linked servers, Endpoints, Jobs, and other Instance level objects under each SQL Server

instance. Likewise, each database is a collection of schemas, tables, users, and other database objects. The hierarchy is shown in Figure 1.3 for your reference.

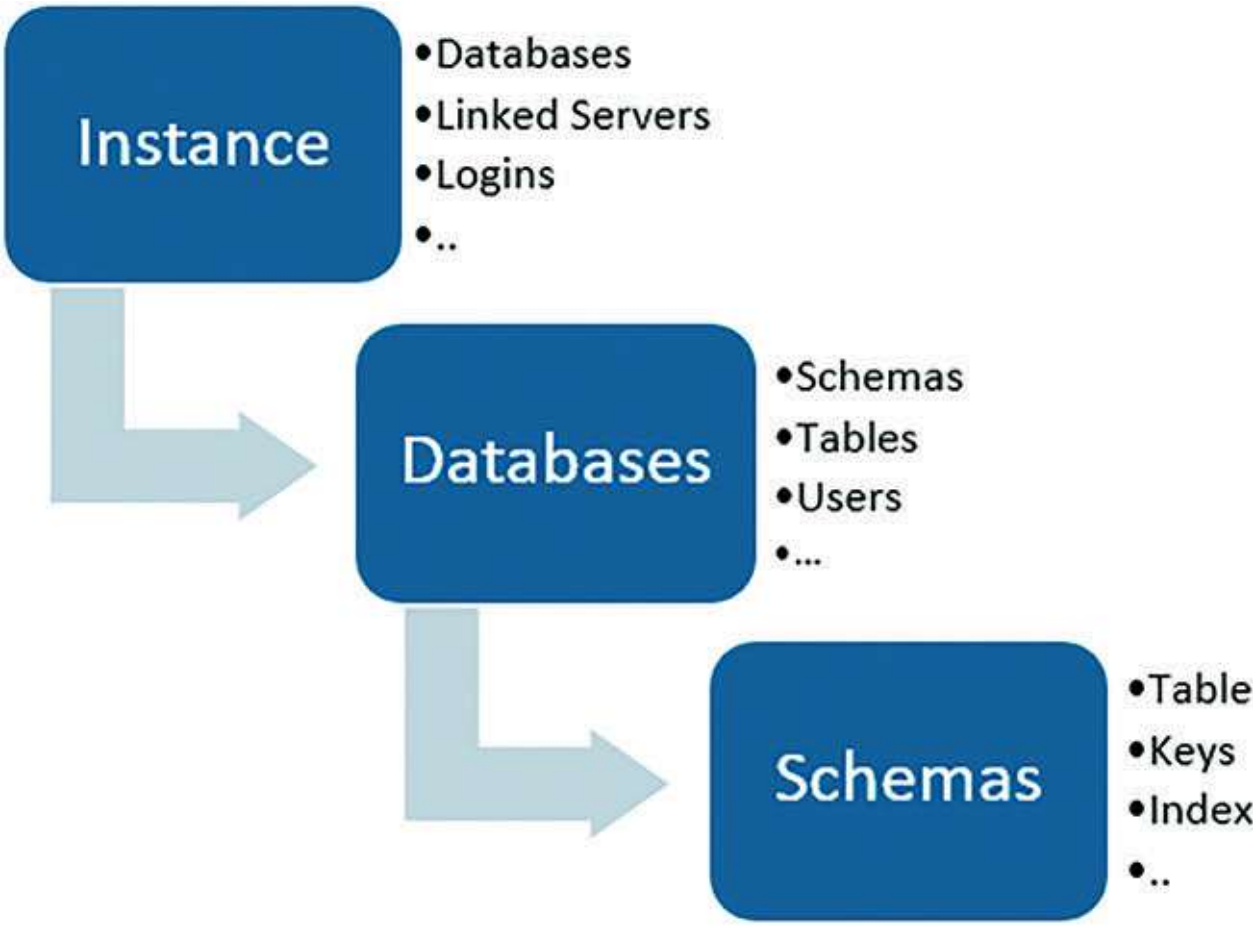


Figure 1.3: SQL Server hierarchy

Using client tools, such as SQL Server Management Studio (SSMS), SQL Server Management Studio (SSMS), SQL Server Management Studio (SSMS, Microsoft Learn or Azure Data Studio (ADS), What is Azure Data Studio - Azure Data Studio, Microsoft Learn, you can connect to SQL Server instances running on Windows or Linux to manage and administer.

A tux icon next to the instance icon in SSMS indicates a Linux-based SQL Server instance, while a plain database icon indicates a Windows-based one.

This helps you quickly identify the operating system of the SQL Server instance you are connected to. See Figure 1.4 for an example.

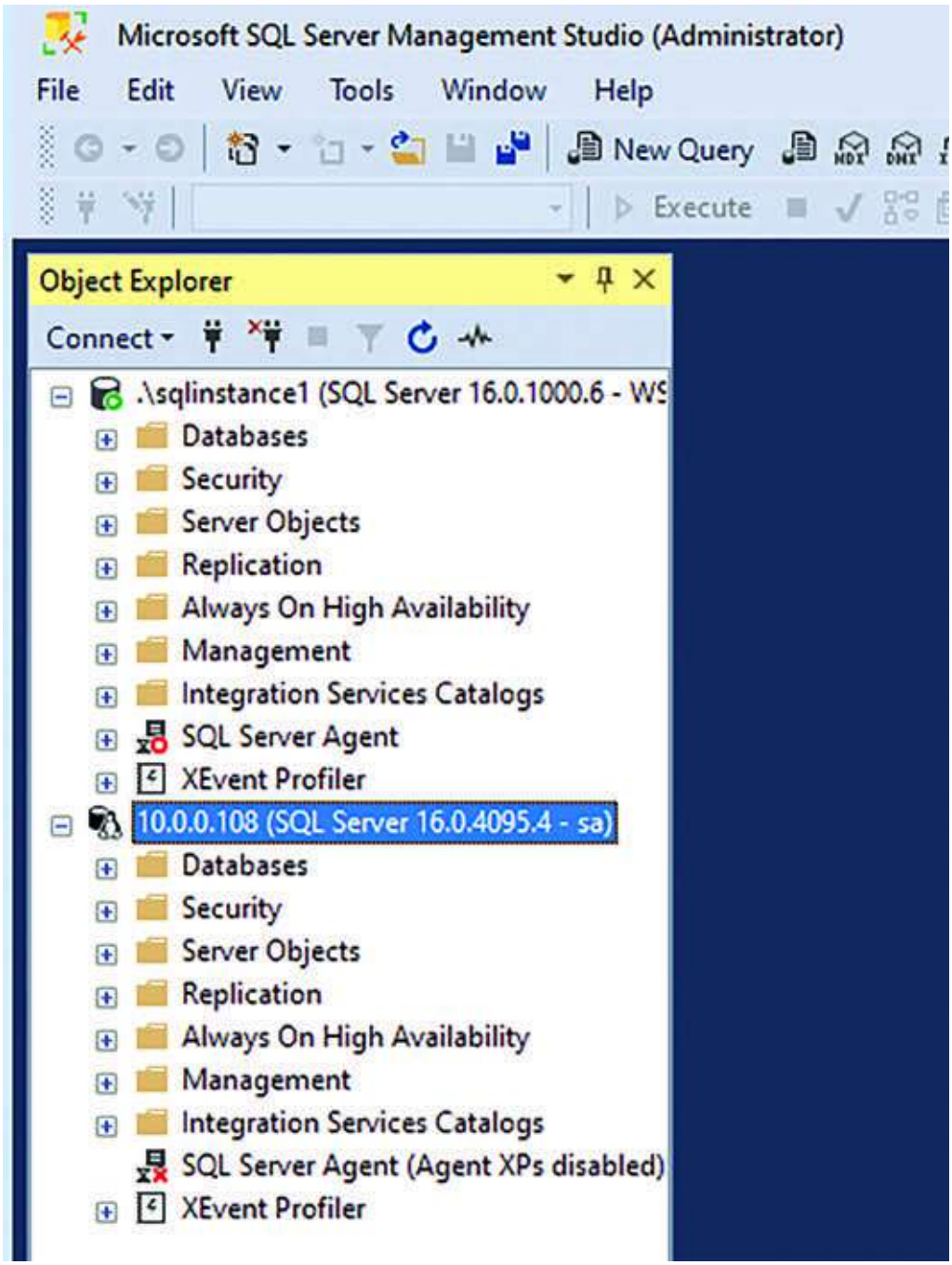


Figure 1.4: Connecting to SQL Server using SSMS 1

Azure Data Studio is a data management client tool that works on Windows, macOS, or Linux. It shows the OS and edition details of the connected

database, as in Figure It also has many extensions to support other databases, such as MySQL, PostgreSQL, and others on cloud or on-premises.

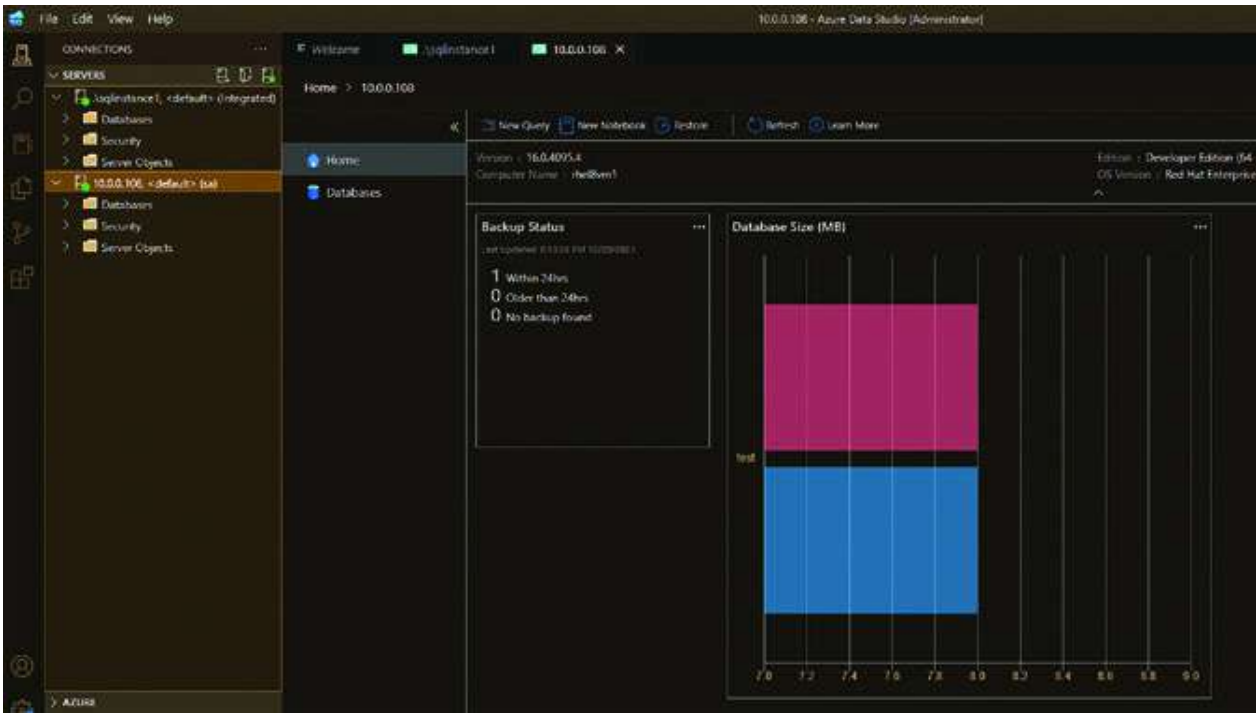


Figure 1.5: Connecting to SQL Server using ADS

SSMS is the preferred tool for SQL Server management and administration, but you may use ADS when you need to connect to different kinds of databases, including SQL Server.

To familiarize yourself with SQL Server and these tools, it is recommended to install SQL Server 2022 Developer edition (free) on Windows by following the instructions provided here: SQL Server installation guide - SQL Server | Microsoft Learn If you prefer Linux, then you can complete the installation of SQL Server 2022 on Linux in less than 2 minutes, on your choice of distribution by following the instructions as documented here: Installation guidance for SQL Server on Linux - SQL Server | ().

SQL Server Internals

Now that you understand SQL Server installation and connecting to them using the client tools, let's learn more about the internals of SQL Server starting with SQL Server databases. In this section, here are the core concepts that we will learn about:

SQL Server Databases- Files and Filegroups

Transaction log architecture

System and User objects

Indexes

Backup and Restore

SQL Server Databases- Files and Filegroups

To understand the internal workings of the SQL Server database, let's start with the basics. When you create an SQL Server database through UI in SSMS, you will notice that a minimum of two files are created on the operating system—a data file and a log file as shown in Figure The data file/s stores the data and the log file is used for logging to help with the recovery of the database.

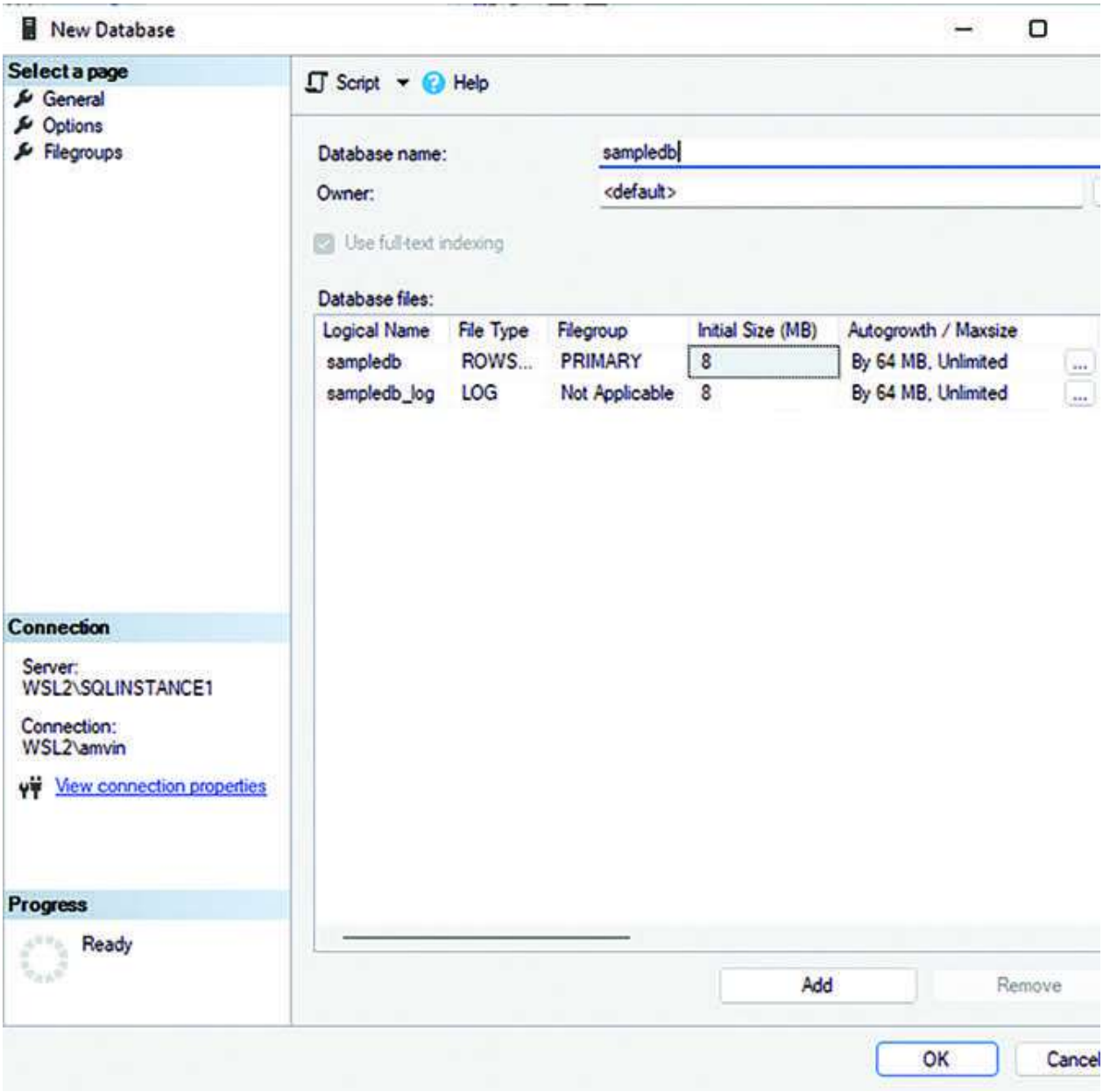


Figure 1.6: SQL Server database properties

Figure 1.7 shows how the database structure looks like when connected to the instance via the SSMS.

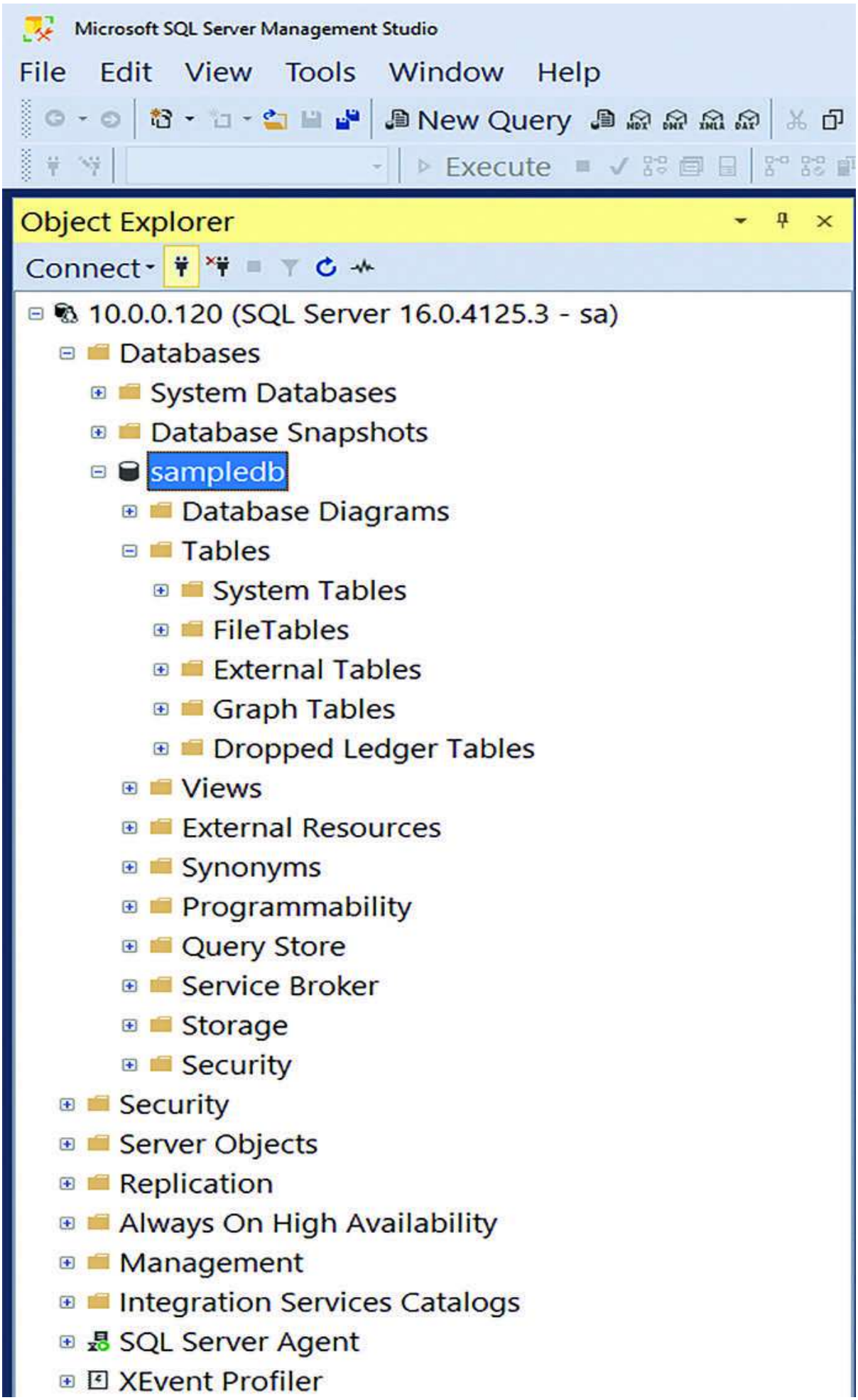

Microsoft SQL Server Management Studio
File Edit View Tools Window Help
New Query
Execute
Object Explorer
Connect
10.0.0.120 (SQL Server 16.0.4125.3 - sa)
Databases
System Databases
Database Snapshots
sampledb
Database Diagrams
Tables
System Tables
FileTables
External Tables
Graph Tables
Dropped Ledger Tables
Views
External Resources
Synonyms
Programmability
Query Store
Service Broker
Storage
Security
Security
Server Objects
Replication
Always On High Availability
Management
Integration Services Catalogs
SQL Server Agent
XEvent Profiler

Figure 1.7: SQL Server database seen in the object explorer in SSMS

You can always create more than one data and log file for a single database. Though it is recommended not to create more than one log file for a database, you can and should always have multiple data files. Every data file is associated with a filegroup; a filegroup is a logical way of grouping data files and they help you spread your data across multiple disks by partitioning your tables and indexes across multiple data files, assisting you with query performance and administrative tasks, such as backup and restore options. Let's see this in action to understand this further:

In the create database sample script here:

We are creating a new database called with three data files: and

Two new filegroups are being created as well: Primary and Secondary.

The primary filegroup has the sampledb.mdf datafile and the secondary filegroup has the other two sampledb1.ndf and sampledb2.ndf files.

We are also making the secondary filegroup as the default filegroup, so every time an object is created inside the sampledb database, it gets created on the files that are part of the secondary filegroup.

All the data files that belong to the secondary filegroup are on the D drive, a different physical disk on the machine, and the data files that belong to the primary filegroup are on the default C drive, ensuring the database I/O is distributed across multiple disks.

The log file is placed on the E drive and it is not part of any filegroup. The E drive is a separate physical disk. This ensures that the Data I/O is separate from Log I/O.

Every file that you create for a database has a logical and physical name. The logical name is what you use in T-SQL commands to refer to the physical names on the operating system. In the following sample script ‘sampledb1’ is the logical file name for sampledb1.ndf which is the physical file on the operating system.

```
CREATE DATABASE [sampledb]
CONTAINMENT = NONE
ON  PRIMARY
( NAME = N’sampledb’, FILENAME = N’C:\Program Files\Microsoft SQL Server\MSSQL16.SQLINSTANCE1\MSSQL\DATA\sampledb.mdf’, SIZE = 8192KB, FILEGROWTH = 65536KB ),
FILEGROUP [secondary]
( NAME = N’sampledb1’, FILENAME = N’D:\sqldata\sampledb1.ndf’, SIZE = 8192KB, FILEGROWTH = 65536KB ),
( NAME = N’sampledb2’, FILENAME = N’D:\sqldata\sampledb2.ndf’, SIZE = 8192KB, FILEGROWTH = 65536KB )
LOG ON
( NAME = N’sampledb_log’, FILENAME = N’E:\sqllogs\sampledb_log.ldf’, SIZE = 8192KB, FILEGROWTH = 65536KB )
WITH LEDGER = OFF

GO
USE [sampledb]
GO
```

```
IF NOT EXISTS (SELECT name FROM sys.filegroups WHERE
is_default=1 AND name = N'secondary') ALTER DATABASE [sampledb]
MODIFY FILEGROUP [secondary] DEFAULT
GO
```

This configuration ensures that the system tables and objects are all in the files that belong to the primary filegroup, and the user tables and objects are in the secondary filegroup. Note that the initial size of the files in the filegroup is same, this ensures that proportionate data is written on the data files that belong to the filegroup. This is a critical aspect of the database design; if you use files of unequal size, then the data will not be balanced, and some files will be used more than others, hence throttling the I/O throughput by causing disk contention. SQL Server uses proportional fill strategy across all files so that when you have files of equal size all the files are used proportionally.

A data file can be part of one filegroup and one database only. Also, a file or a filegroup can never be shared amongst databases; they are exclusively part of one database only.

Let us now understand the nuances of Data I/O. SQL Server mostly uses random I/O for data files and sequential I/O for log files. When we say data I/O, it refers to the process of reading and writing data pages from and to the disk specifically onto to the datafiles. A page is the fundamental unit of data storage in SQL Server. The size of a page is 8KB. A page holds user and metadata. Eight physically contiguous pages is called as an Extent. Thus, the size of an extent is 64 KB (8 pages × 8 KB/page) as depicted in Figure

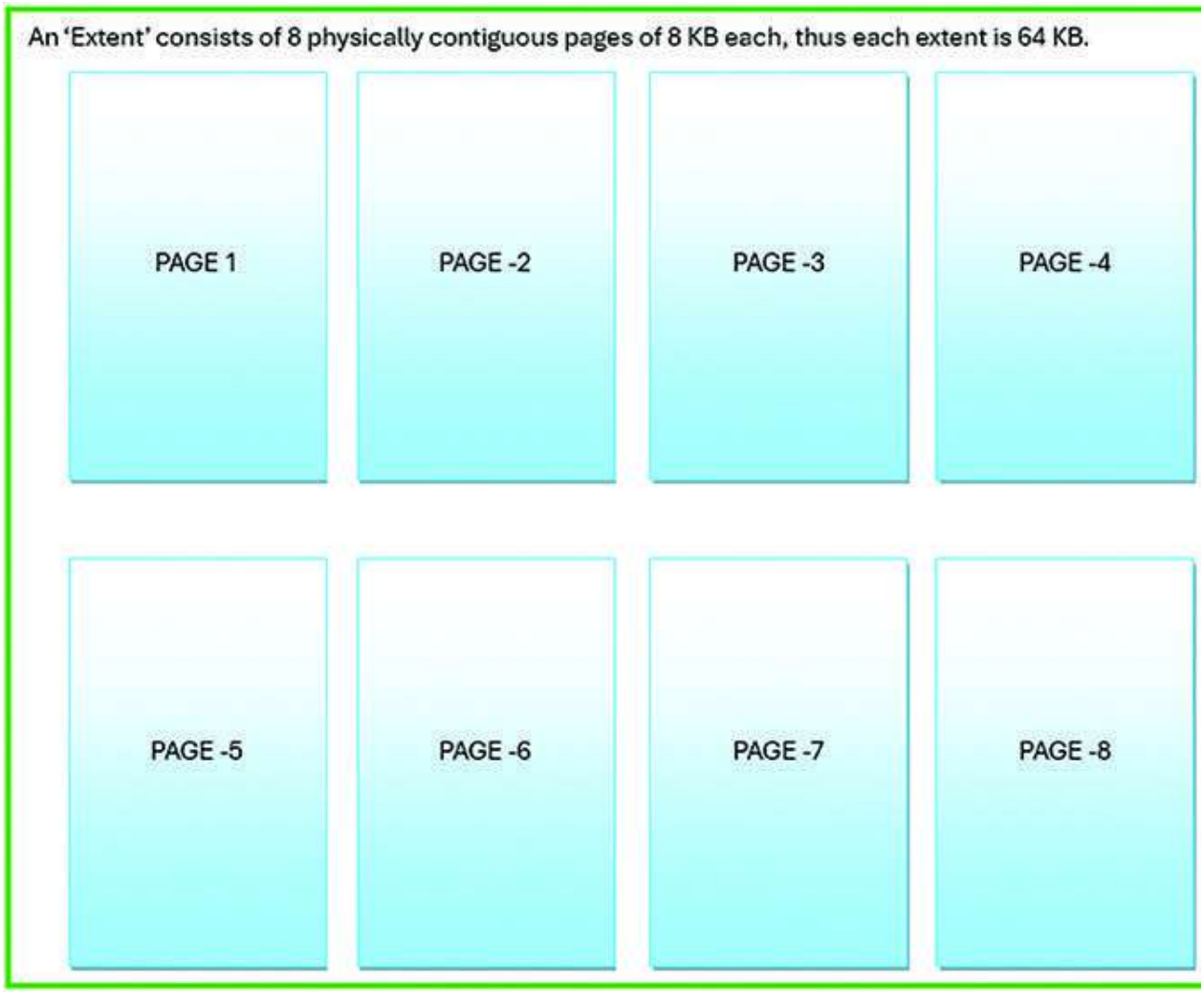


Figure 1.8: A pictorial representation of an Extent and page

An extent is used as the basic unit for space management by SQL Server. There are two types of extents:

Mixed The pages part of this extent belongs to different objects, in fact, each page could belong to different objects, hence one mixed extent can be shared by up to 8 objects.

Uniform All the eight contiguous pages in this extent belong to the same object.

SQL Server uses allocation maps, also referred to as system pages, to record allocation of extents to data objects. There are two types of allocation maps:

Global Allocation Map GAM pages record what extents have been allocated. Each GAM covers 64000 extents. If the bit is 1, then that extent is free, else, it is allocated.

Shared Global Allocation Map (SGAM): SGAM records extents that are currently used as mixed extend and have at least one unused page. Each SGAM covers 64000 extents as well. So, for each extent, there is a bit, and if that is set to 1, it means that it is a mixed extent with at least 1 page as free. If the bit is 0, then the extent is not a mixed extent or if it is a mixed extent, it does not have a free page.

So, using the GAM and SGAM allocation maps makes it simple for SQL Server to identify mixed extents that have one free page or identify uniform extent to allocate to an object. While this is helpful at extent level, there is also a metadata page called Page Free Space (PFS) pages. After an extent is allocated to an object, the database engine uses the PFS pages to record which pages in the extent are allocated or free. This helps the SQL engine to allocate a new page needed to insert a new row or index key values.

When you query system dynamic management views (system-created objects that you can refer to monitor, manage, or tune SQL Server performance) a few DMVs that you can use to query and learn more about pages and extents are:

sys.database_files

sys.dm_db_file_space_usage

Here’s an example showing you how to use them:

```
select db_name (database_id) as db_name, file_name (file_id) as File_id,
total_page_Count, allocated_extent_page_count as
pages_in_allocated_extent, unallocated_extent_page_count as
pages_in_unallocated_extents
, mixed_extent_page_count as pages_in_mixed_extent,
modified_Extent_page_count as
pages_modified_in_allocation_extent_since_last_full_backup   from
sys.dm_db_file_space_usage
```

Figure 1.9: SQL Server page details

```
select file_name(file_id) as filename, size as size_in_pages, (size*8) as
size_in_MB  from sys.database_files where type = 0
```

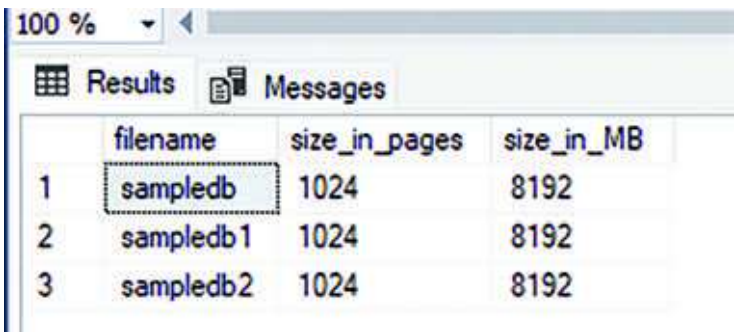


Figure 1.10: SQL Server database Size in MB

To learn more about the pages and extent architecture, it is recommended that you refer to the official Microsoft documentation: Pages and Extents Architecture Guide - SQL Server | Microsoft Learn

Transaction Log Architecture

SQL Server Log file is an essential component required for the databases to be recovered to a consistent state. The transaction log records all the transactions and modifications made by every transaction. A transaction log is a string of log records. Each log record can be identified through the unique Log sequence number (LSN). The LSN of the current log record will always be greater than the previous log record. Thus LSNs follow sequential series. General recommendations about transaction log are:

You should try and have only one transaction log file per database, unlike the data files where it's recommended to have multiple data files.

For better I/O throughput, always create the log files of the databases on separate drives from the data files as the I/O pattern for Log files is completely different from data files.

Try and ensure that you pre-size your log file to maximum size that you think the log file will grow to so that this will avoid unnecessary log file growth during production hours.

After reading the following section, you will have a better idea on the aforementioned recommendations.

Microsoft SQL Server uses the Write-ahead logging (WAL) technique to ensure the durability and consistency of the database after a restart. The WAL protocol guarantees that no data modifications are written to the disk before

the associated log records are written to the disk. When you make changes to the database, such as creating a table, index or inserting/updating a row, the page that contains the row is fetched from the disk to part of the memory (cache) called buffer cache (also known as buffer pool), if it's not there already in the buffer pool. Once fetched in the buffer pool, the page is latched (not locked) and modified; this page is now referred to as dirty page.

The page also has the information on the transaction log record that modified the page. The activity of writing from memory to disk is called flush. For every modification, a transaction log record is inserted in the log cache. First, the log cache is flushed and then the corresponding buffer caches are flushed to the disk. The operation that performs the flush of data pages from the buffer cache to the disk is called as checkpoint.

Imagine a scenario, where a lot of modifications are being performed on a database marking multiple pages as dirty, the log records for those modifications have already been flushed to the log file and the data pages were being flushed, when the system goes down, without the data buffers flushed completely. When the database restarts and performs recovery, it can now refer to the log file to identify the active transactions, then if it were committed redo (roll forward) those transactions to ensure the changes are persisted. And undo (rollback) the transactions that were not committed. This ensures the databases is consistent before and after the restart.

Let us look at the transaction logs and their physical architecture in detail. The physical SQL Server log file is divided into several virtual log files (VLF) as shown in Figure 1.11 which is an example of the database sample db created based on the T-SQL script shared in the previous section. The sizes of the virtual log files are decided by SQL Server dynamically while the log file is created or extended based on the auto-growth setting. The SQL Server database engine tries to keep the VLFs as few as possible. SQL Server uses

the VLFs as a mechanism to manage and reuse the physical log file of the database.

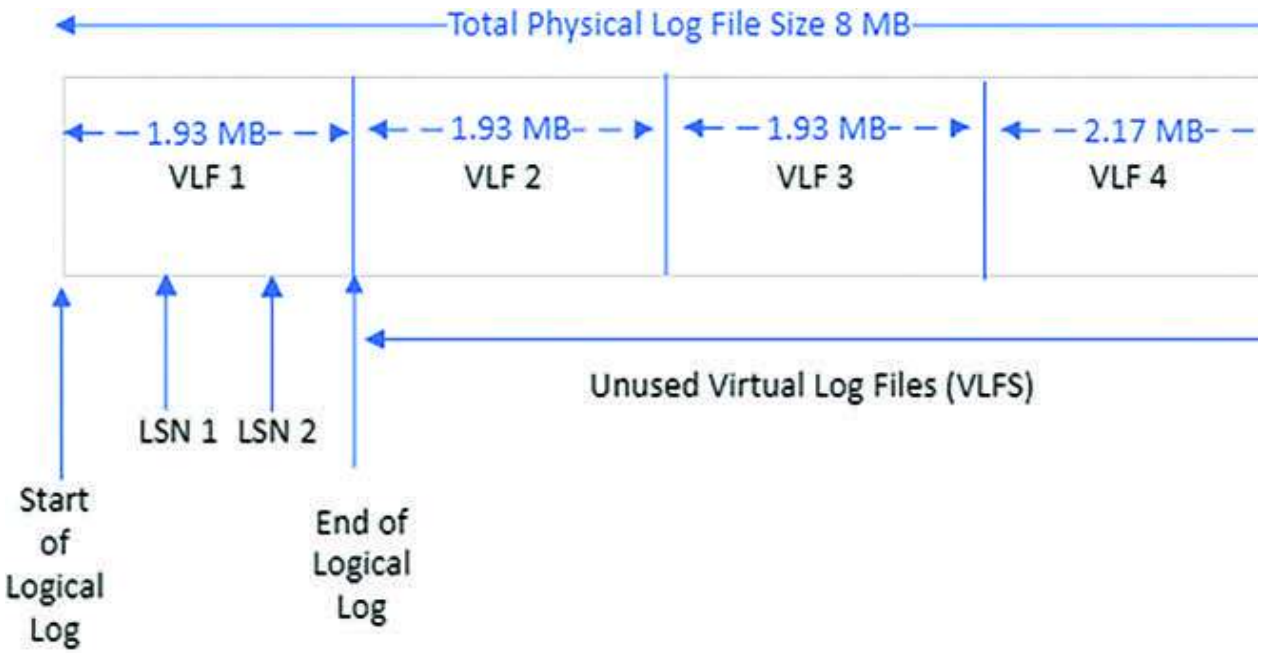
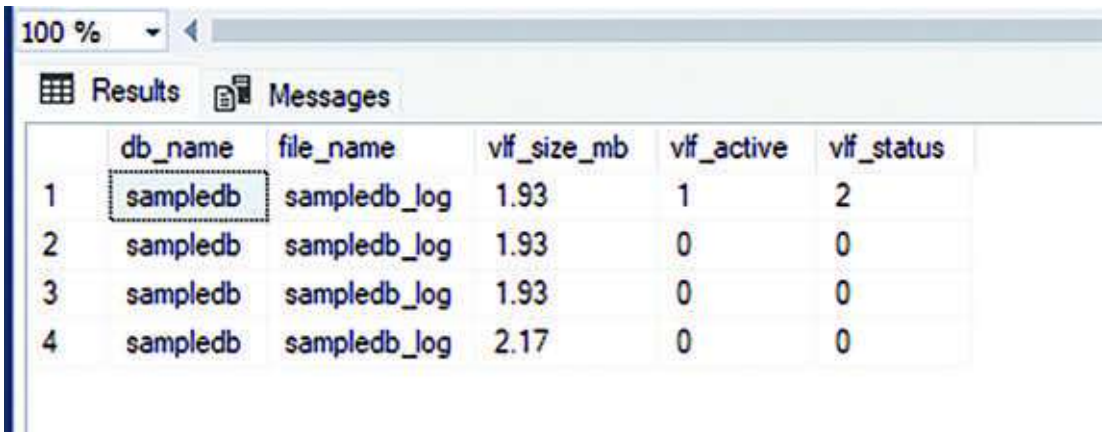


Figure 1.11: Transactional Log Logical Architecture

You can also query this using the DMV: sys.dm_db_log_info with the query as shown here:

select db_name(database_id) as db_name, file_name(file_id) as file_name, vlf_size_mb, vlf_active, vlf_status from sys.dm_db_log_info (8)



	db_name	file_name	vlf_size_mb	vlf_active	vlf_status
1	sampledb	sampledb_log	1.93	1	2
2	sampledb	sampledb_log	1.93	0	0
3	sampledb	sampledb_log	1.93	0	0
4	sampledb	sampledb_log	2.17	0	0

Figure 1.12: Query SQL Server Transactional log

There is a total of 4 VLFs and the size of each VLF is shown in the table. As you can also see, only the first VLF is active, the rest of the VLFs are unused or inactive as the logical log ends at the end of the virtual log file.

Now, as modifications happen in the database, you will notice that the logical log starts to increase and the VLFs change the state from inactive (unused) to active based on the next output after running the same query as shown here:

select db_name(database_id) as db_name, file_name(file_id) as file_name, vlf_size_mb, vlf_active, vlf_status from sys.dm_db_log_info (8)

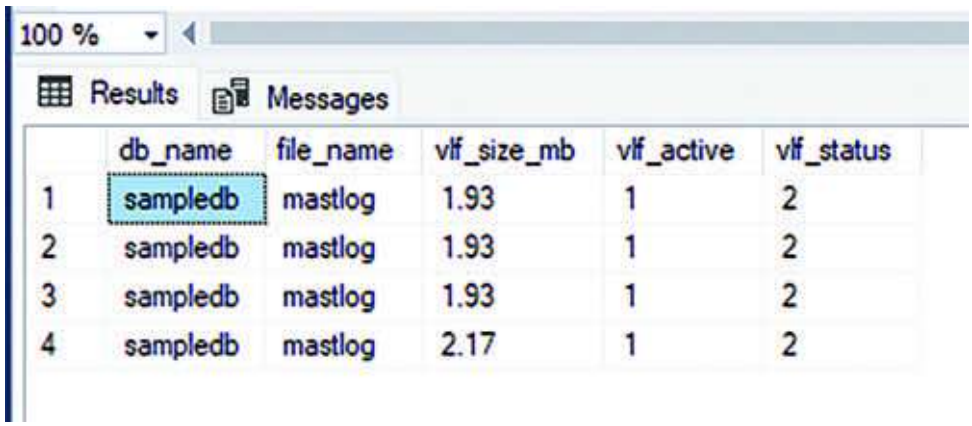
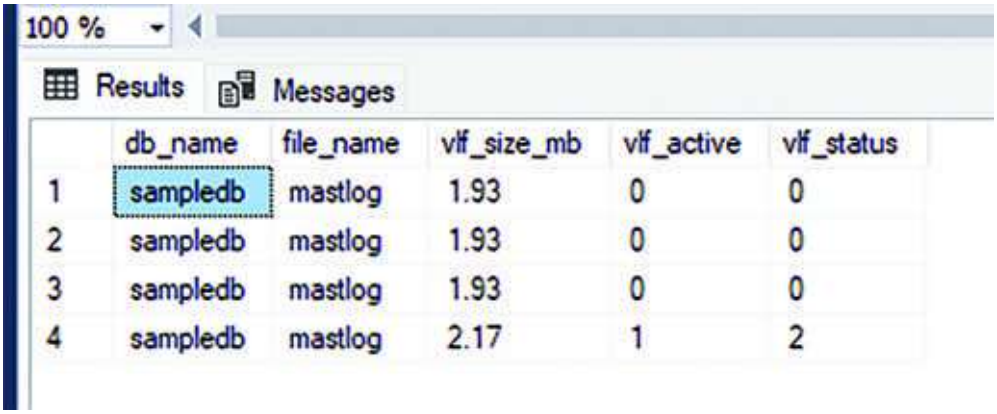


Figure 1.13: Query SQL Server Transactional log

As you can see, all four VLFs are now active, meaning, the entire log file is in use. We will now run a checkpoint, causing the flushing of the log cache and corresponding buffer cache which will truncate the log file for reuse without the need of the growth. Here is the output of the sys.dm_db_log_info and as you can see the VLFs 1, 2, and 3 are now inactive and the last VLF is the only active VLF.



	db_name	file_name	vlf_size_mb	vlf_active	vlf_status
1	sampledb	mastlog	1.93	0	0
2	sampledb	mastlog	1.93	0	0
3	sampledb	mastlog	1.93	0	0
4	sampledb	mastlog	2.17	1	2

Figure 1.14: Query SQL Server Transactional log

The file architecture, as shown in Figure after the log truncation, marks the first three VLFs as inactive. Note that the transaction log file did not grow but it wrapped around shown by the green line and marked the first VLF as inactive, meaning ready to be used. Also, for easy understanding, it is shown the last checkpoint, the start of the VLF and the min LSN are the same but in a real environment the min LSN could be behind the checkpoint LSN as that transaction might not have been completed when the checkpoint occurred.

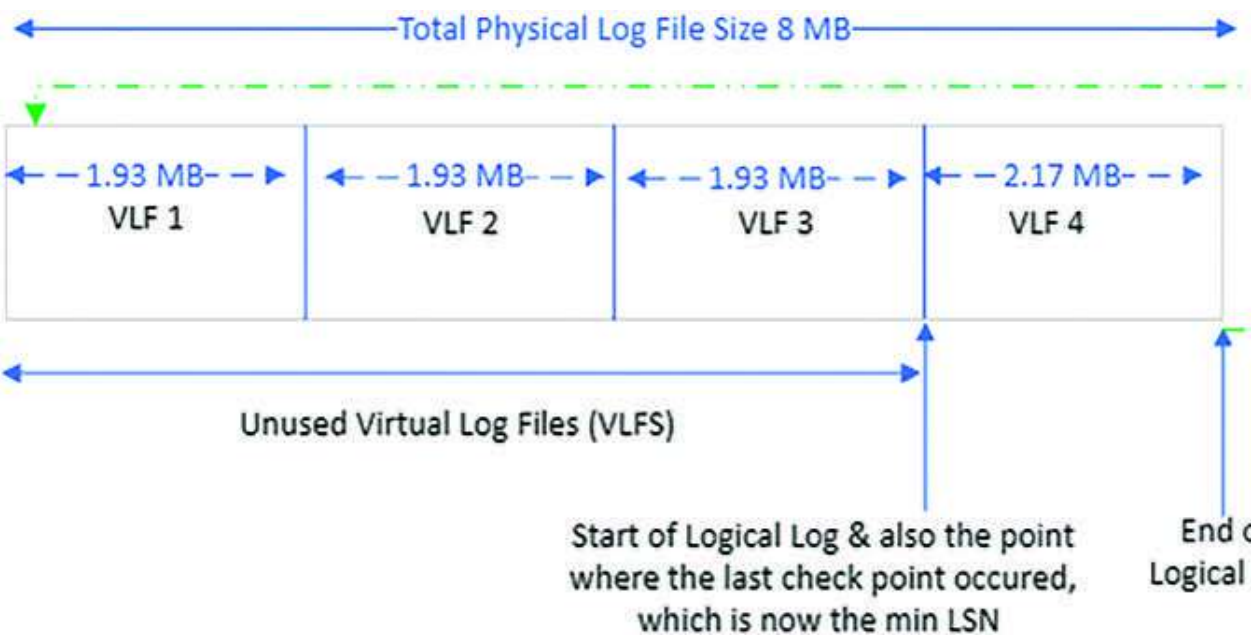


Figure 1.15: Transactional log logical architecture

Truncation of the log is the process that marks a VLF as an inactive signalling SQL Server that it can use for logging as all the changes done previously on that VLF have been successfully captured in the database and can be recovered consistently. Log truncation ensures that log files eventually do not fill all the disk space and provides the ability to reuse the virtual log files. Also, remember a VLF cannot be partially active or partially inactive. If there is a single log record that is still active, the entire VLF is marked as active.

The log truncation occurs automatically after the following events:

When the database is configured under Simple recovery model, after a checkpoint log truncation occurs.

When the database is configured under Full recovery model after the log backup is completed and a checkpoint has occurred since the previous backup, then the truncation occurs.

If you see a lot many VLFs being created and the log file size increasing, that means that the truncation of the log file is not happening. To find the reason why the log truncation is not happening for a database, you can refer sys.databases and use a query like:
select name as db_name, log_reuse_wait, log_reuse_wait_desc from sys.databases sd where database_id= db_id(sd.name)

This query should let you know the reason why the log_reuse is not happening. A few common reasons are: Log backup, Active transaction, and

Replication. To see the entire list, refer to the official Microsoft documentation and refer to the columns: log_reuse_wait_desc sys.databases (Transact-SQL) - SQL Server | Microsoft Learn

Finally, when talking about data files, we spoke about data pages and extents. Data pages are normally 8KB in size and the extents is a collection of 8 pages, so the size of an extent is 64 KB. Similarly, the log block consists of log records. The log records themselves can vary in size but are always in integer multiple of 512 bytes, which is the minimum sector size that SQL Server supports. The maximum size of the log block is 60 KB. So, as you can understand, the log flushes are small in size and mostly sequential I/O. To learn more about the transaction log and its architecture, read the official Microsoft documentation available here: SQL Server transaction log architecture and management guide - SQL Server | Microsoft Learn

System Objects and Users Objects

When you connect to a SQL Server instance, you will notice that there are a few system objects, such as the system databases, tables, views, logins, and other objects that are pre-created. You can list all the system objects contained in the schemas named sys or INFORMATION_SCHEMA using the query:

```
select * from sys.system_objects
```

System objects store the metadata, that is, data about data. It is essential for SQL Server functionality. Most of these objects are physically stored in the resource database, one of the system databases. All these system objects logically appear in the sys schema of every database including the user databases. These system objects help in the functioning, monitoring, and administration of the SQL Server.

Let's first look at the system databases that are by default created as soon as you install SQL Server on Windows or Linux. There are 5 system databases, and you can view 4 out of the 5 system databases when you connect to the SQL Server instance as shown in Figure For Azure SQL Database and elastic pools, you will see only the master and tempdb database. For the Azure SQL Managed instance, Azure SQL Server VMs all the system databases apply.

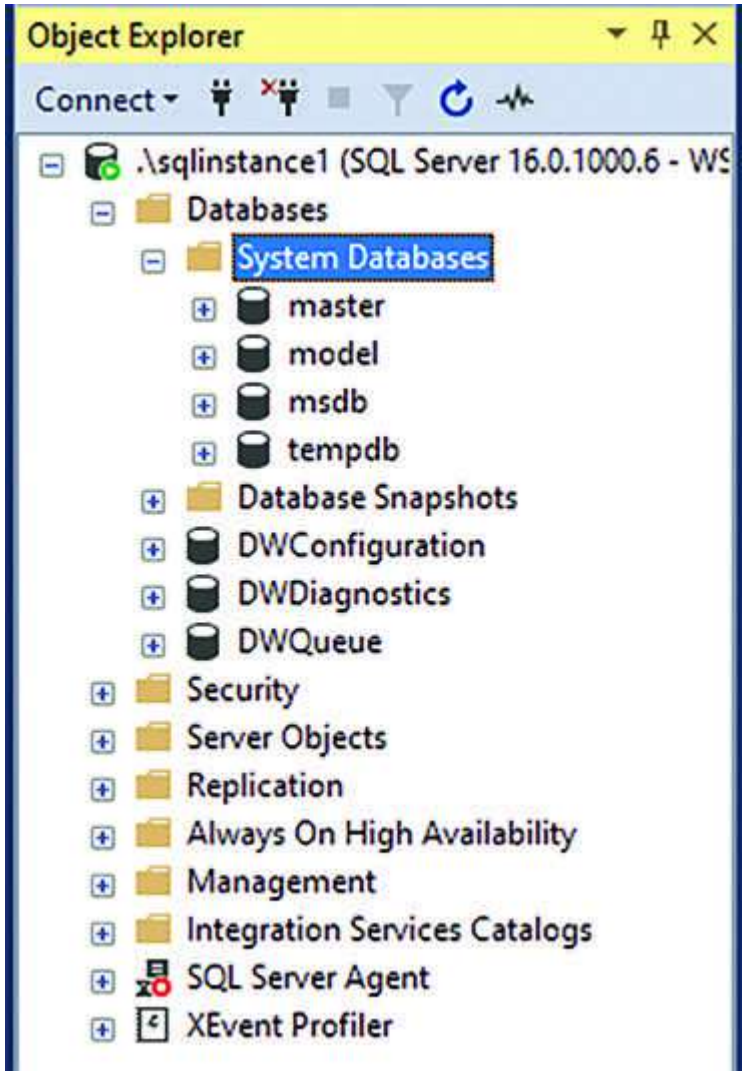


Figure 1.16: System databases as seen from SSMS

Master This database is used to record instance-level metadata objects, such as login accounts, endpoints, Linked servers, service master key (SMK), and other system configurations specifically related to

Model This is the database that is used as the template to create other databases on the instance. Most of the database options like the recovery model and other contents of the model database are copied to the new database that is created.

Msdb This system database is specifically used by the SQL Server agent for scheduling, tracking, and monitoring the various jobs and alerts that are configured. This contains system objects that can be used to track the backup restore history, agent job run history, dbmail, and other features provided by SQL Server.

Tempdb Database: This is a global resource available to all users connected to an instance of SQL Server. This database is used only to store temporary objects, and hence every time the SQL Server is restarted, the tempdb is recreated. All the previous data in tempdb will be lost after restarting the SQL Server. You should not use this database to create objects that you intend to persist after SQL Server restart.

Often, this database is also used internally by the SQL Server engine to create temporary objects for storing intermediate results of a query, creating work files for joins, or storing intermediate sort results or version data. You can track the usage of the tempdb using the Dynamic Management views, such as:

sys.dm_db_file_space_usage

sys.dm_db_session_space_usage

For sample queries please refer: tempdb database - SQL Server | Microsoft Learn

Since, this is a global single database and is used across the entire SQL Server instance, to avoid performance throttling on this database, there are

published recommendations and guidelines for capacity planning and optimizing the tempdb performance. In fact, there have been multiple performance improvements in the tempdb in SQL Server releases. Some of the major features introduced were setting up tempdb during the installation of SQL Server and memory-optimized tempdb metadata in SQL Server 2022 as well there are improvements to achieve better concurrency for the system pages. For details please see: tempdb database - SQL Server | Microsoft Learn

Resource This is a read-only database and contains all the system objects that are included with SQL Server. You cannot view this database like the others when you connect to the SQL Server instance. But, all the system objects are physically persisted in this database and all these objects appear in the sys schema of every database that is created.

You can also verify in the errorlog : up The physical files of the database for SQL Server on Windows is located at: Files\Microsoft SQL and for SQL Server on Linux, it is included with the sqlserver.sfp files located at It is recommended and suggested not to move these files to a different location, as every time SQL Server is upgraded to a newer version, the system objects are updated by copying the new resource database file at the same location, which makes the upgrade process simple. Earlier, the system objects had to be dropped and recreated which was a cumbersome task.

User Objects

You've learned how to install SQL Server and client tools, connect to the instance, create database, and look at the system databases and other system objects. Now, let's learn about the various types of tables that you can create to store your data in SQL Server and how you can optimize retrieval of the stored data.

In SQL server when you expand the tables option underneath the database, you see six different types of tables that you can create when working with SQL Server 2022 version as shown in Figure

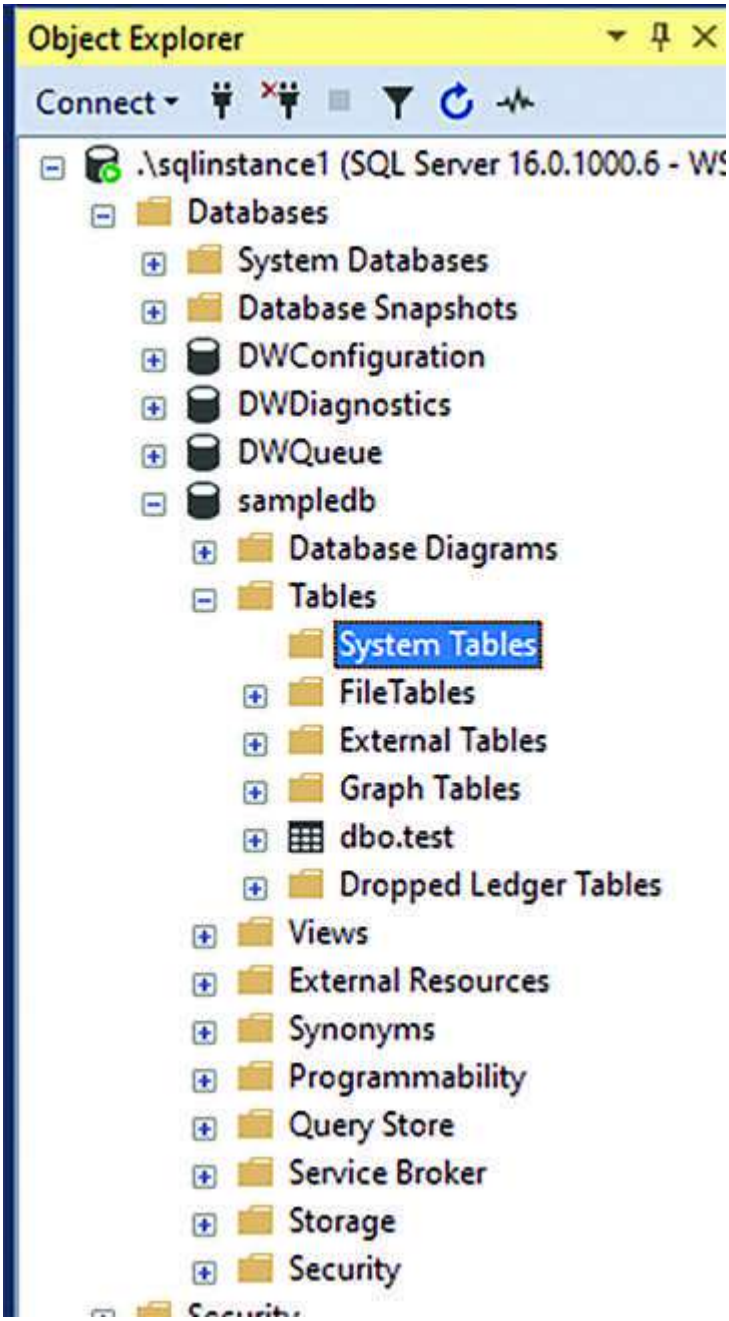


Figure 1.17: Different Types of tables as seen from SSMS

System We already know that the system tables store the configuration of the server and have metadata. The data within the system tables are made available through the system views and hence cannot be updated or queried by the users.

The SQL Server is normally associated with handling structured data, while the SQL Server can also be used to handle unstructured data. Filestream is one such feature in SQL Server that enabled applications to store unstructured data, such as images, documents, and videos on a file system at the same time provide transactional consistency between the unstructured data and corresponding structured data.

When you enable Filestream, you can create Filetables that let an application integrates its storage and data management components and provide features, such as full text and semantic search over unstructured data and metadata. As of now, at the time of writing this book, this feature is only supported for SQL Server on Windows and not for the SQL Server instances deployed on Linux.

To learn more about filestream, see FILESTREAM (SQL Server) - SQL Server | Microsoft Learn

External These tables let you access data from other databases or storages, such as Oracle, Teradata, Azure Data Lake Storage, S3 compatible object storage, or Azure Blob storage using SQL Server. This is called data virtualization, which you can enable with the Polybase feature. It works for SQL Server on Windows and Linux.

To create and use an external table, you need to do three steps. First, create a database scoped credential with the access rights to the data source. Second, create a data source that uses the credential. Third, create the external table that uses the data source. For more information on

Polybase and how to use it, refer: Introducing data virtualization with PolyBase - SQL Server | Microsoft Learn

Graph Tables: You can also enable the Graph database capabilities for SQL Server allowing you to model and query data in a graph format. To work with Graphs in SQL Server, you can create the node and edge tables and then using the basic T-SQL query syntax and the MATCH clause, you can do pattern matching and multi-hop navigation through the graphs. These are also supported for both SQL Server on Windows and Linux.

To learn more about Graphs and how you can get started with them in SQL Server, please refer : SQL Graph Architecture - SQL Server | Microsoft Learn

User These are the normal user tables that you create using the T-SQL syntax of “create table”. Their data is logically organized in a row-and-column format. Each row inserted into this table is a unique record and each column is a field in the record. SQL Server also provides you with the ability to assign properties to the table and to each column in the table that control the data that is allowed. These are called constraints. For efficient retrieval of data, SQL Server allows you to create indexes and statistics on these tables that help you seek or scan data from the tables in a fast and efficient manner.

You can also compress the data that is stored in the tables to optimize storage. To improve the manageability of huge tables, you can also create partitions on the tables called partitioned tables that allow you to partition data horizontally across multiple filegroups in a database. Thus they allow you to access subset of data quickly and efficiently.

Normally, the data in the user tables are stored in row format, but, if need be, you can also change the storage to columnar format by creating columnstore indexes, which then stores the data in a column format that provides better performance for analytics workload. Tables with columnstore indexes are called

Columnstores are generally preferred for data analytics or data warehouse workloads. To learn more about columnstores refer: Columnstore indexes: Overview - SQL Server | Microsoft Learn (

Memory Optimized The data in these tables is all stored in-memory and hence there is no requirement of fetching data pages from disk to cache. The complete data of these tables all time is stored in-memory. You have a choice of creating durable or non-durable memory. A memory-optimized table, when durable ensures it, meets all transactions requirements; they are atomic, isolated, and consistent, meaning when the database is restarted, all the data is persisted and not lost. The checkpoint process in these tables is different from regular disk-based tables; here you have checkpoint files (data and delta file pairs), which are used for recovery.

The transaction log though is the same as that used in disk-based tables. During recovery, the transaction log and checkpoint files are used to recover the memory-optimized tables after restart.

You would normally use the memory-optimized tables when you need high performance for OLTP workloads, which is also required to process large numbers of inserts concurrently. Read more about memory-

optimized tables here: Introduction to Memory-Optimized Tables - SQL Server | Microsoft Learn (

Dropped Ledger Tables: These are tables that are created when you enable the new SQL Server 2022 Ledger feature for specific tables and then drop the tables. When the tables are dropped after the ledger feature is enabled, they are logically dropped and removed from the schema but physically the table is renamed and shows up under the dropped ledger tables section. To learn more about ledger feature, please Ledger overview - SQL Server | Microsoft Learn

Indexes

The previous section focused on tables, and how tables are differentiated based on the storage format. In this section, our focus is on indexes. Figure 1.18 shows the various indexes that can be created in SQL Server based on the primary storage type of the table.

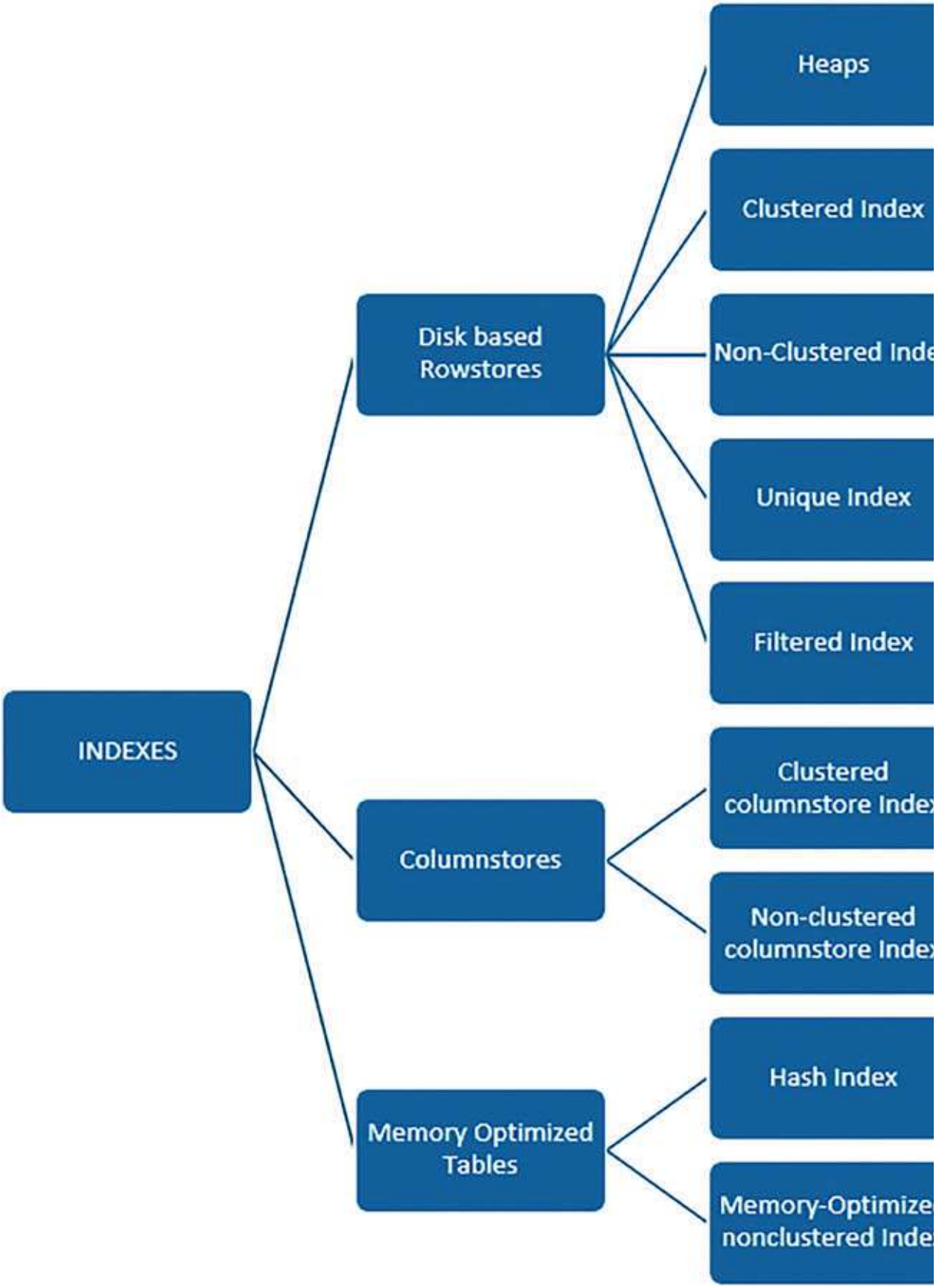


Figure 1.18: Different Types of Indexes in SQL Server

Before we look at each of these indexes, let us understand what they are and why they are needed.

A table stores data for us, but just storing data is not enough when you are building a data layer; you also need the ability and capability to retrieve the data from tables quickly and efficiently, for your application functionality. This is where indexes come in. They are data structures that help to speed up the retrieval of data (rows or columns) from tables. They help in reducing the disk I/O by efficiently seeking the data (row or column) that is required by the application from the table.

When you execute a query T-SQL like: product_name, product_quantity, customer_name from sales.orders where

Here the column names that we want to retrieve are: The table name is orders which is in the sales schema and the data we specifically retrieve is for the order_id

The SQL Server query optimizer evaluates methods to retrieve this data quickly and efficiently. The options that the optimizer has depends on the table structure and presence of any indexes. For example, if the table has a clustered index on then it can quickly traverse the index and efficiently seek order_id 7 and retrieve the required data. If the table does not contain any index, then a scan of the table will be performed, which means all rows or most of the rows in the table are read and then we extract the data that meets the criteria of the query. Generally, a table scan consumes more time and resources when compared to an index seek or scan.

Hence, it is critical you ensure that the right indexes and table structures are created when working on tables so that you can get the best from your SQL Server.

Now, that you understand how query optimizer works and how indexes could help, let's look at the various indexes starting with disk-based row stores.

Disk-based Rowstore tables

Data stored without specifying any order is called heap. A table without any clustered index is a heap. You can create non-clustered indexes on a heap.. The data is usually stored in the order that the rows are inserted into the table.

Heaps are usually recommended when you are creating staging tables, a type of temporary holding table that gets its data as part of an insert from another data source. As there is no ordering enforced during the inserts, they are faster. A heap is used for smaller tables that are mostly scanned, and you don't run queries that seek specific rows from the table.

Clustered When you create a clustered index, the data in the entire table is sorted and stored based on the key that was chosen as the clustered index. You can only have one clustered index per table because you can order the data in a table in only one way at a given time. The data structure used for clustered and non-clustered indexes is a B+ tree (Balanced Tree). The B+ tree consists of a root page, which is the top node of the structure and then the intermediate or the branch nodes which have points to the leaf nodes, the bottom nodes containing the data pages.

Figure 1.19 shows how the clustered leaf level has the data itself and how the traversal of the index tree happens when I want to search order_id SQL

Server starts from the root node, because 7 is less than 10, it immediately finds the branch node page with key=7 and in turn, finds the pointer to the leaf level that has the order id as 7, and then at leaf level, we get the data associated with the key

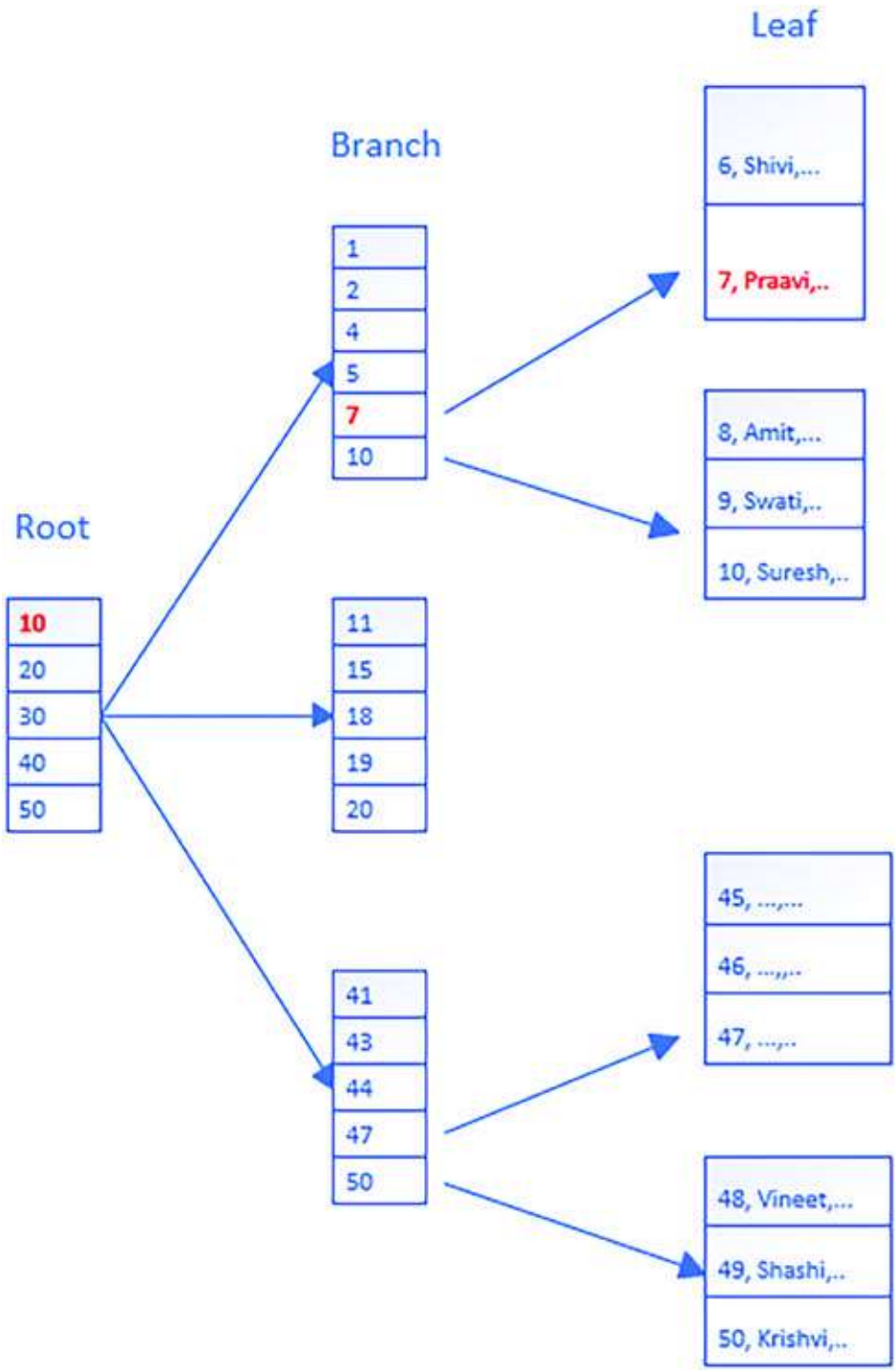


Figure 1.19: B-Tree Structure

As we are using the B+ tree structure, the leaf nodes are a doubly linked list, making it easier for traversal between the leaf nodes. When a clustered index is created on a heap which already has non-clustered indexes, then as part of the clustered index creation, all the non-clustered indexes should also be rebuilt to contain the clustered index key values instead of the row identifiers (RID). It is generally preferred to build the clustered index first and then build the non-clustered indexes. It is recommended that you read the create clustered index T-SQL syntax to learn more: Create Clustered Indexes - SQL Server | Microsoft Learn

Non-clustered A non-clustered index like clustered index, uses the B+ tree. It has a separate structure from the data rows. But, unlike the clustered index, the non-clustered index contains the key values, and the key value has a pointer to the data row called the row locator at the leaf level. If the non-clustered index is created on a heap, then this row locator is a pointer to the row that matches the key value, for a clustered table, the row locator is the clustered index key. Then the clustered index scan is performed to reach the data. Hence, as you might have noticed the non-clustered index requires an additional lookup to retrieve the data from the table of clustered index.

With non-clustered indexes you also can add non-key columns to the leaf level of the non-clustered index, this helps you execute fully covered queries. When you add the non-key columns to the non-clustered index leaf level, they are called the included columns. Do refer to the T-SQL syntax to learn how to create non-clustered indexes. Create Nonclustered Indexes - SQL Server | Microsoft Learn

Unique Index: You can use constraints in SQL Server to enforce rules on the tables. For example, you might want to ensure that certain columns have

unique values and avoid the insertion of duplicate values. You can enforce this on a table by creating a unique constraint. When you create a unique constraint, it does allow the value null in the column/s only once. This constraint is enforced on the table by SQL Server by creating a unique non-clustered index by default. For syntax on the creation, please refer: Create Unique Indexes - SQL Server | Microsoft Learn

Filtered Index: This is a type of non-clustered index that creates the index structure only on a portion of rows in the table. When creating this non-clustered index, you basically define a predicate, and the index is created on the data that satisfies the predicate. This type of index is beneficial when you have a table which is queried most often on a well-defined subset of data. For example, you might have a table with more than two decades of data, but most of the time your application only queries for orders that are received after the year 2021. For such a scenario, you can create a filtered index on that table for data that is after the year 2021. This way the index structure is smaller, making it easier to maintain and saving storage space as you create the index structure only for data after 2021 and not for the previous 20 years. For the create T-SQL syntax, refer: Create filtered indexes - SQL Server | Microsoft Learn

Columnstore Tables

SQL Server gives you the ability to store data in tables as row format is mostly preferred for Online Transaction Processing (OLTP) workloads or as columnar format which is preferred for data warehouse or Online Analytics Processing (OLAP) type of workloads. The columnstore also gives an additional benefit of compressing the data almost up to 10 times when compared to the original data size. If you want the table to save data in columnar format, then you need to create a clustered columnstore index (CCI).

Clustered Columnstore Index As soon as you create a CCI on a table, the process of changing the physical storage of the entire table from rowstore to columnstore starts. When the CCI is created the rows are grouped in Rowgroups, each rowgroup contains a maximum of 1,048,576(~1 Million) rows. These rowgroups are compressed and physically stored in columnar format. For the rowgroups to be closed and ready to be compressed, they must have at least 102400 (~100K) rows. This ensures the least fragmentation when converted to columnstore.

But, if a rowgroup does not have about 100K rows, then the rowgroup remains in an open state and is in rowstore format on the disk, which is also called delta rowgroups. Only when the threshold of 100K or the maximum 1 Million rows are reached, does the rowgroup transition to closed state and then compressed. This happens in the background through the tuple-mover thread.

A collection of delta rowgroups across the table is called deltastore. Remember these delta stores have data in a rowstore format and use a B-tree based clustered index which is used by the columnstore indexes. So, when a query is run against a columnstore table, the query optimizer combines the results from both the data in columstore format and the result from the deltastore.

Non-Clustered Columnstore Index A NCCI and CCI both when created on a table, convert the table from rowstore to columnstore. The major difference between the NCCI vs. CCI is that CCI is normally the primary index created on the table that is stored in columnar format. But you can create an NCCI as a secondary index on a rowstore table that might also have a B-tree-based Clustered index. This means that when you create the NCCI on a table, the

same data is available now in both rowstore and columnstore format. When you run a query against a table that has an NCCI on the table, the query optimizer identifies that if this is an analytical query, then NCCI is used but if this is a normal OLTP query, then B-Tree based index is used to generate the result of the query. Hence, an NCCI provides you with the ability to run real time operational analytics with minimum changes; all that is needed is to create NCCI on your current table and get started with real time operational analytics.

With NCCI, you can create filtered indexes to choose which part of the table to store in columstore and which part in rowstore. You can also use the compression delay options to decide when to switch the table to columnstore.

To learn more about the columstore indexes, read: Columnstore indexes: Overview - SQL Server | Microsoft Learn

Memory Optimized Tables

The primary storage for memory-optimized tables is the main memory, that is, when the transactions are run on these tables the rows are read and written to the main memory and not the disk. If you have created a memory-optimized table that is set as durable, then a copy of the table data is maintained on disk, for only durability. The transactions are processed against the table that is in memory.

Since the table is in memory, there are no data pages or extents on disk. All the data rows are in memory, and it is the indexes that connect the rows together in memory. Hence, it is cardinal and a requirement for memory-optimized tables to have at least one index. There are two types of indexes that you can create on memory-optimized tables; they are Hash and Memory-optimized non-clustered Index, also called as Range index.

Hash A hash index is also an in-memory structure; it is an array of pointers, and each element in the array is called a hash bucket. The bucket is 8 bytes in size and is used to store the memory address of a link list of key entries as shown in Figure This figure is taken from the official Microsoft documentation that is available here: SQL Server and Azure SQL index architecture and design guide - SQL Server | Microsoft Learn

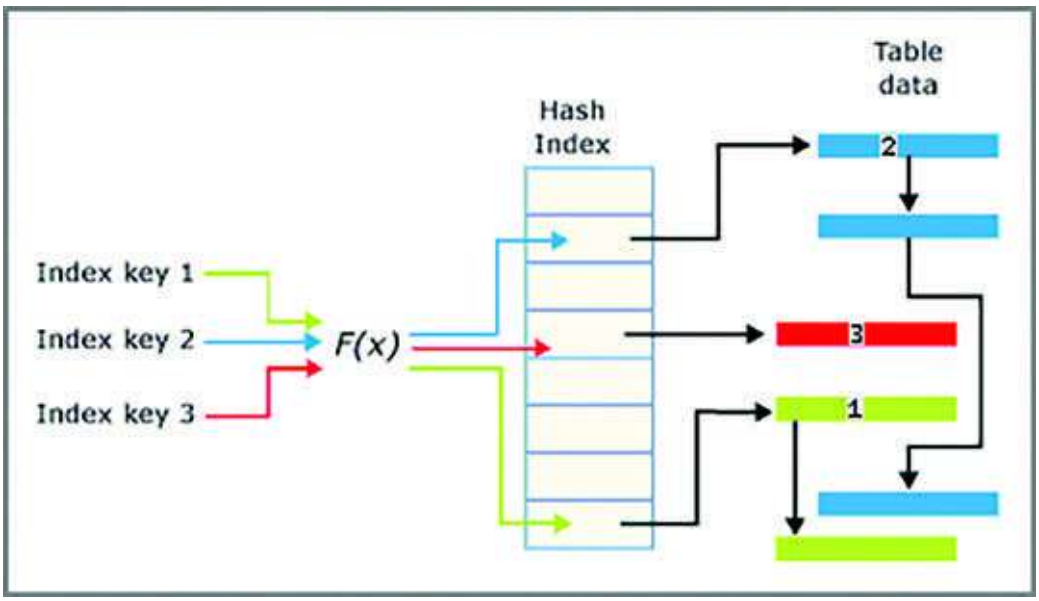


Figure 1.20: Hash Index with buckets

The function where x is the key column value on which the hash index is created, is used to generate a hash of the key value, and then based on the hash generated, it gets allocated to one of the buckets in the hash index, which then stores the pointer to the memory address to the actual data row that resides in memory. If the hash is the same for one or more key values, then each table data entry points to the next entry in a link list of entries, all chained to the current hash or bucket. If two index keys are mapped to the same hash bucket, that is called as hash The longer the link list of table rows,

it deteriorates the performance of the query. Hence, when you are choosing to create a Hash index on an in-memory table, try and choose the column in the table that has unique values. When creating the hash index on a table, you need to provide the number of buckets; the number you specify is rounded up to the next power of two by the SQL Server database engine and that many hash buckets are created.

If you create too many hash buckets, then memory is wasted, and chances are not all the hash buckets will be utilized. Remember, each hash bucket is sized at 8 KB! If you choose too small a number, then you end up with more hash collisions which impacts the query performance. The general recommendation is to create buckets between 1 and 2 times the number of distinct values in the index key. Overestimating the bucket count is always better than underestimating.

Memory Optimized Non-Clustered This is also called as Range index. If you are not sure of how many buckets to create for a hash index or the workload consists of queries that normally search for a range of key values, then you can opt to create the Range Index. This index uses the Bw-Tree structure, described in this whitepaper published by Microsoft: bw-tree-icde2013-camera-ready.dvi (microsoft.com)

In general, the Bw-Tree is like the B-tree that we looked at earlier for non-clustered indexes. Each index page contains a set of ordered key values and pointers. The pointers leaf to lower-level index pages or data rows at the leaf level. If multiple data rows have the same key value, then like hash indexes, multiple data rows are linked together.

In a Bw-Tree there is a page mapping table, that has the address mapping to the index pages, and then using the index page, you traverse to the leaf of the page which finally has the key value and the address to the data row in

memory. This Figure 1.21 is from a whitepaper https://download.microsoft.com/download/8/3/6/8360731A-A27C-4684-BC88-FC7B5849A133/SQL_Server_2016_In_Memory_OLTP_White_Paper.pdf published by Microsoft that explains In-memory tables and index structures; we are using it for quick reference, and if you are interested to know more, go through the white paper.

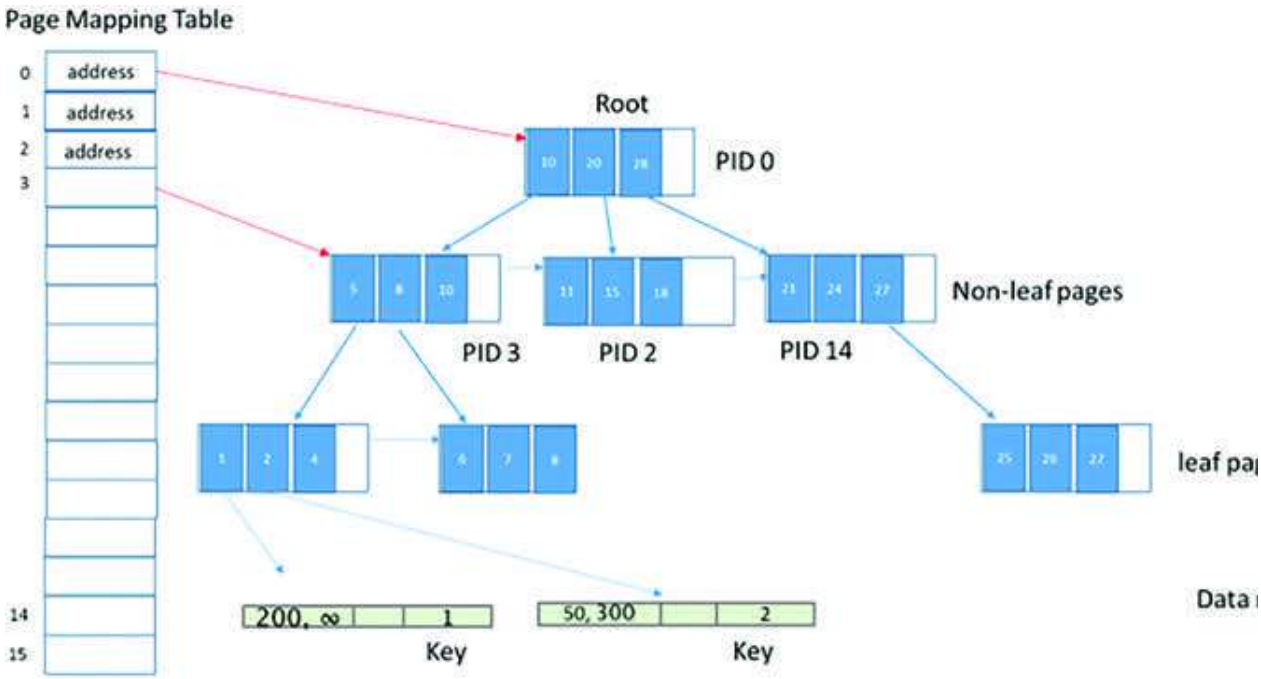


Figure 1.21: BW-Tree structure

Apart from all these indexes that we discussed, there are other indexes, such as Full-text, XML, and spatial Indexes which are all special indexes used for sophisticated word searches in character strings, for XML data type columns or for geometry data types respectively.

If you intend to learn more about this, we recommend you to refer: Indexes - SQL Server | Microsoft Learn for details.

Backup and Restore

Business continuity is one of the critical aspects of any database design. Backup and restore is an important tool in your arsenal to work towards your business continuity plan. We have a chapter solely on business continuity dedicated to helping you understand the various options across the entire suite of SQL products that you can rely on for safeguarding your data.

You've got to plan, create, and importantly test your backup restore strategy to ensure that you are ready to avoid and minimize any catastrophic data loss. When you are testing your backup and restore strategy, the test cases should cover a variety of situations ranging from page restores to restoring from media failures to an entire data estate restoration across data centres.

In this section, we are going to explore the different types of backups and the restore scenarios that are enabled with those backups. Once the database is restored, the next step is the recovery of the database, which is a process of bringing the database to an online state that is consistent with what the database was like before the failure or corruption.

Before we look at the types of backups and restores, it is essential for us to understand that backup and restore operations occur within the context of a recovery model of a database. The recovery model is a database setting that affects the following aspects of a database:

It determines how the transactions are logged in the transaction log

It enables or disables transaction log backups depending on the recovery model chosen.

It supports different types of restore operations.

There are three recovery models: full, simple, and bulk-logged. If you do not choose a recovery model when you create the database, it will use the full recovery model by default. This is because the database inherits the settings from the model system database which is by default set to full recovery mode.

When you set the database to a simple recovery model, all transactions are logged but do not allow log backups, because the logs are truncated after a checkpoint as discussed in earlier sections. As changes since the most recent backup is unprotected, if there is a crash, you have to redo those transactions. If you want to avoid transaction log management and want to simplify backup and restore, a simple recovery model is your choice, provided the business runs smoothly even with the data loss or can redo transactions that were not part of the backup.

In the full recovery model, all transactions are fully logged, the log truncation only happens when you take a log backup. Thus, log backups are allowed, giving you the ability to restore to any point in time, provided you have complete backups up to the point of failure. It supports almost all the restore options. If you choose to give importance to minimum work loss exposure at the cost of administrative overhead, then you should choose a full recovery model.

This is more of a supplement to the full recovery model. In the full recovery model, all the transactions are fully logged, but if you have bulk inserts happening, then you can switch the database to use the bulk logged recovery model, which ensures the transactions are minimally logged for bulk operations. Thus, it reduces the log space usage and provides high performance for bulk operations in SQL. Log backups are allowed, size of the log backup is mostly bigger than the normal log backups as it includes the minimally logged operations. Point-in-time restores are not supported.

For more details on the recovery model, read Recovery Models (SQL Server) - SQL Server | Microsoft Learn

Types of Backups and Restores

The Figure 1.22 summarized the different types of backups and restores. Not all are supported across all the databases; as mentioned the backups and restores supported depends on the recovery model and the edition of SQL Server being used.

There are three types of backups that you can take in SQL Server:

Database Backup (Full)

Log Backup

Differential Backup

Figure 1.22: Different types of Backups and Restores in SQL Server

Database Backup

When you run a backup, you normally specify the object that you are backing up, which could be the database, a filegroup, log, and you also specify the backup media which is the destination, where the backup file will be created. The media could be a local drive, network drive, a cloud storage like backing to Azure storage, or any other S3 based object storage.

A database(full) backup, backs up the entire database; it represents the whole database once the backup is finished. In the simplest form, the backup T-SQL syntax is:

“Backup Database to disk = ‘/var/opt/mssql/log/backup/db.bak’ “

Here, we are using ‘.bak’ but you can choose the extension of your choice, the general standard for ease of reference is ‘.bak’ for database backups, ‘.trn’ for transaction log backups, and ‘.diff’ for differential backup. Again, this is not a rule, but a general practice followed across. Also note, that the sample shown here is for a Linux-based SQL Server deployment, the path of the target could also be a Windows path.

When taking the database backup, you could also specify a specific filegroup/file to be backed up, also known as file backup. The filegroup that you specify could be readwrite or readonly. The filegroups you specify are only the ones that are backed up, a sample T-SQL syntax looks like this:

“ Backup Database filegroup= to disk = ‘/var/opt/mssql/log/backup/filegroup_name.bak’ “

Similarly, you could also take a partial backup of the database, which by default only backups the primary filegroup and all the readwrite filegroups of the database. You can also backup the readonly filegroups, but you will have to specify the readonly filegroup names that you intend to backup.

“ Backup Database READ_WRITE_FILEGROUPS, read_only filegroups> to disk = ‘/var/opt/mssql/log/backup/dbfilename.bak’ “

Full database backups copy the whole database, so they take longer and need more space as the database grows. Therefore, you should use a combination of full, partial, filegroup, differential, and log backups, depending on the recovery model of the database, which should be full, or bulk logged.

When you run backups, you could also specify various other parameters to enhance the speed and backup size, such as max transfer size, block size, compression, encryption and more. For a complete list, see: BACKUP (Transact-SQL) - SQL Server | Microsoft Learn

Log Backups

Log Backups enable you to take the backup of the transactions log and capture all the changes since the last log or full backup. Minimally, you must have at least one full backup before you create log backups.

For example, if you have taken a full backup and followed it up with a log backup, the log backup captures all the transactions that were not captured by the previous full backup. Log backups, as mentioned earlier, are supported for full and bulk-logged recovery models only.

The T-SQL sample syntax for log backup is:

Backup log to disk = '/var/opt/mssql/db_name.trn' "

If for any reason, the transaction log is damaged, work that is performed since the last valid backup is lost, hence remember to put your log files on fault-tolerant storage.

Similarly, if a database is damaged or you are in the process of restoring a database, then it is recommended that you create a tail log backup, which captures any log records that have not been backed up. The tail log backup can be taken in the following scenarios:

The database is online, and you plan to perform a restore operation on the database, you can take the tail log backup using the norecovery option, example:

“Backup log db_name to disk= ‘/var/opt/mssql/tail.trn’ with norecovery”

The database is offline or fails to come online with messages like then you can use the no_truncate option, sample T-SQL command is:

“Backup log db_name to disk= ‘/var/opt/mssql/tail.trn’ with no_truncate”

If the database is damaged, then you can also try taking the tail-log backup using the continue_after_error option, for example:

“Backup log db_name to disk= ‘/var/opt/mssql/tail.trn’ with continue_after_error “

For details on log backup syntax, please refer: Transaction log backups - SQL Server | Microsoft Learn

Differential Backups

A data backup captures the changes since its base backup. This base backup could be the last full or partial backup. Thus, a differential backup can only be taken once a base backup which is a full or partial backup or a filegroup backup was previously taken. Remember the differential backup only contains the differential data since the last base backup.

Differential backups are backups that generally finish faster when compared to full backups and hence used as a supplement to the base backup. Generally, the backup strategy consists of taking full backups once a week or twice a week, then followed by more frequent differential backups that capture changes and finally the log backups which are generally taken minutes away to back up the transactional log. To know more, refer: Differential Backups (SQL Server) - SQL Server | Microsoft Learn

Copy-only backups

When you take a backup, it changes the database and affects how later backups are restored. But sometimes, you need the ability to take a one-off backup that does not affect the later backups, maybe a scenario where you want to take the current database and restore for tests on another server, you can issue a copy-only backup, that is independent of the sequence of conventional backups.

You can take a copy-only full backup or log backups. Remember with copy-only transactional log backups the transactional logs are not truncated like it does with normal log backups when the database is in full or bulk logged recovery model. The T-SQL sample for taking a copy only backup is as simple as given:

Backup database to disk = ‘/var/opt/mssql/backup/db.bak’ with copy_only

And a log backup with copy_only command would look like

Backup log to disk = ‘/var/opt/mssql/backup/db.trn’ with copy_only.

Finally, the time intervals and type of backup is all dependent on the recovery point objective and recovery time interval that is set by your organization. We discuss this in detail in the business continuity chapter of this book.

Restores:

Once you have the valid backup files, restore is a multi-phase process that copies the data and log pages from the specified backup files to the database and then performs recovery on the database to bring it online.

In a restore process, you can use multiple backup files to restore the database, for example, a full backup, a differential backup, and a couple of transaction log backup. If you intend to restore the complete database, with the assumption that the recovery model is set to Full, here are the steps that are generally followed:

Initiate a tail log backup of the current database; this ensures that you copy the latest changes that are not yet backed up in any of the backup files you have.

You then restore the full database backup with the norecovery option which allows you to restore any other backup file that you have.

You then restore the differential backup and all the log backups including the tail log back taken in step 1, for the last backup file that is restored, you enable the recovery option to ensure that the database goes online without any issues.

As shown in Figure you can perform various types of restores to list a few:

A complete database restores the entire database, starting with restoring a full database backup, followed by the other backups of the database that are available, which could be a differential backup or log backups. For this entire operation, the database is offline and comes online only after a successful restore. Refer to a sample T-SQL syntax here: Complete Database Restores (Full Recovery Model) - SQL Server | Microsoft Learn

“Restore database from disk = ‘C:\backups\db.bak’ with recovery “

If you have more backups to restore, you should use the norecovery option instead of the recovery option. This will keep the database in a restoring state until you apply the last backup and bring the database online.

A file restore, restores a file or filegroup in a multi-filegroup File restores can be performed in online or offline mode, if the database is online, for which you intend to restore the file, the filegroup to which the file belongs that you intend to restore remains offline.

When performing an online file restore, it is recommended to take the tail log backup and then restore the file and filegroup in norecovery mode and then restore the tail log backup to ensure the database is consistent. Example syntax is shared below, for details refer: File Restores (Full Recovery Model) - SQL Server | Microsoft Learn

Restore database file= from disk = file> with followed by tail log backup restore.

A page restore is available in full or bulk-logged recovery models You can perform the page restore in online mode. To know more refer: Restore Pages (SQL Server) - SQL Server | Microsoft Learn T-SQL sample shared:

“Restore database page= from disk = file> with followed by log backup restores with recovery.

A piecemeal restore operation restores the database in stages, starting with the primary and the secondary This is normally used when you have multi-filegroup database backups. The syntax is specified using the partial option as shown here:

“Restore database read_write_filegroups from disk = file> with partial, recovery”

Since, you have recovered the database, it is online now, but can’t access those files that have not been recovered like the read_only filegroups. You can then also restore any read_only filegroups using the file restore syntax shared earlier. You need to use the recovery option. For details refer: Piecemeal Restores (SQL Server) - SQL Server | Microsoft Learn

A transaction log restore works only for databases with full or bulk-logged recovery The transaction log restore gives you the ability to perform point-in-time restore as you can reach your desired recovery point. The syntax is shared here; for details refer: Apply Transaction Log Backups (SQL Server) - SQL Server | Microsoft Learn

“ Restore log from disk =log backup file> with norecovery “,

Once you have reached the last log file backup that you intend to restore you run the same command with a recovery option to bring the database online.

To create a backup and restore strategy for your databases, you need to make sure that you run and test the strategy for every situation. If you don’t test these strategies, you may not know how to do it when it is crucial, because you have not practised enough. Also, another critical aspect is documenting the restore steps and ensuring your team members have access to it, and that each member is aware of their responsibilities.

Introduction to SQL Server 2022

At the time of writing this book, SQL Server 2022 is the newest version available and the best SQL Server release for Azure integration so far. If you want to create a Hybrid environment that can work on any platform, whether it is a different operating system, On-premises or in cloud, on virtual machines or containers, then SQL Server 2022 is the platform for you.

To see all the new features of SQL Server 2022, you can check the official Microsoft documentation that shows every feature that came with SQL Server 2022. Here is the link, What's new in SQL Server 2022 - SQL Server | Microsoft Learn

Based on my understanding, Microsoft SQL Server 2022 focuses on five critical pillars and the most attractive features under each of those pillars are shown in Figure This figure does not show all the new features but only the most significant ones. Let us look at each of these pillars and respective features, some of which are in preview.

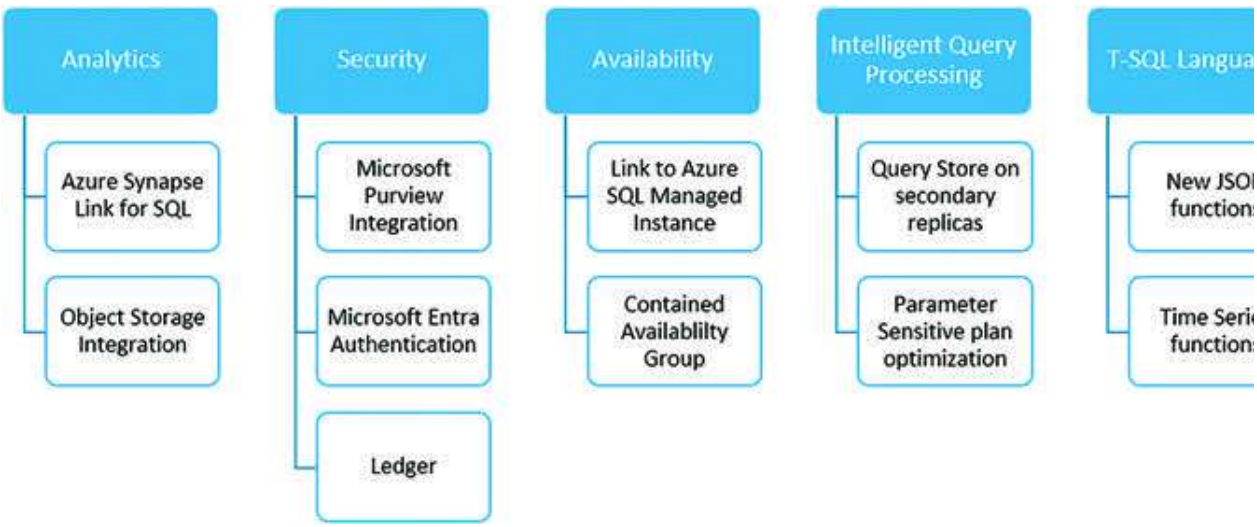


Figure 1.23: SQL Server 2022, listing few latest features

Analytics

Azure Synapse Link for SQL Server 2022:

Earlier in this chapter, we talked about Non-Clustered columnstore Indexes (NCCI) in the indexes section. They are useful for real time operational analytics, as they let you run the analytics workload on the same system that runs the transactional workload. This way, you don't need to create ETL (Extract, Transform, and Load) Jobs for real time analytics.

With SQL Server 2022, you can also copy your operational data to an Azure Synapse Analytics dedicated SQL pool from SQL Server 2022 almost instantly, with little impact on the operational data and no ETL jobs. SQL Server 2022 lets you create a link connection that connects a SQL Server 2022 and an Azure Synapse Analytics SQL pool easily. When you create the link, you choose the source database that sends the data to the Azure Synapse dedicated SQL pool. You can select or deselect tables as you wish. A key point to remember is that when the link is activated, it transfers the full initial load from your source database and then continues to transfer the incremental changes using the change feed feature in SQL Server 2022, which reads the transaction log to find the latest changes and send them to Synapse.

You can stop or pause the link, but there is a difference. When you stop the link and restart it, the full initial load is transferred again, but when

you pause and restart, only the incremental changes that were not transferred when the link was paused are transferred. To transfer the changes from the source to the target Azure Synapse dedicated SQL pool, there is a temporary stage called landing zone, where the data is moved from SQL Server 2022 to a landing zone and then from the landing zone to the Azure Synapse dedicated SQL pool. You need an Azure data lake storage Gen2 account to make the landing zone.

This feature is useful for situations where you need to do near real-time analytics on your operational data with little effect on the transactional workloads. If your organization permits data storage in secure cloud environments like Azure, you can use this feature. With some simple clicks and settings, you can access a complete and advanced analytics platform for your analytics workload.

To know more on how to configure the Azure Synapse Link for SQL Server 2022, please refer: Create Azure Synapse Link for SQL Server 2022 - Azure Synapse Analytics | Microsoft Learn

Object Storage Integration:

One of the new features introduced with SQL Server 2022 is the ability to connect to any S3-compatible object storage, such as MinIO, NetApp, Nutanix, Pure Storage, and more. SQL Server supports both Backup/Restore and data lake virtualization with Polybase integration.

Polybase integration gives you the ability to read or write data into these S3-based object storages with ease using T-SQL and treat data outside SQL Server as external tables that you access by creating credentials. The object storage integration is supported for both SQL Server on Linux & Windows. The only difference is that SQL Server on Windows supports

both Basic authentication and Pass-through authentication, but SQL Server on Linux at the time of writing this book only supports Basic authentication.

You can read a blog post about how to use backup and restore and data lake virtualization with S3 object storage for SQL Server containers on Kubernetes: SQL Server containers on Kubernetes with S3-compatible object storage - Getting started - Microsoft Community Hub

Security

Microsoft Entra Authentication for SQL Server:

SQL Server 2022 introduces a new authentication method with Microsoft Entra ID (formerly Azure Active Directory). Microsoft Entra ID is Azure's cloud-based identity and access management service. You can always federate on-premises based Windows Server Active Directory with Microsoft Entra ID on Azure, thus providing a centralized repository to manage access to your resources across deployments.

Microsoft Entra Authentication is supported by Azure SQL Databases, Azure SQL Server managed instances, SQL Server on Windows Azure virtual machines, and Azure Synapse analytics and now if you have SQL Server 2022 hosted outside of Azure, you could use Microsoft Entra Authentication to authenticate to SQL Server. It is based on the OAuth 2.0 authorization framework.

To enable Microsoft Entra Authentication, you need to connect the non-Azure based SQL Server 2022 instance to Azure using the Azure Arc. Once your Azure Arc enables your SQL Server, then you can turn on the Microsoft Entra Authentication feature. There are multiple authentication methods supported by Microsoft Entra ID and they are:

Default Here you authenticate through non-interactive mechanisms which is a passwordless authentication. It is mostly used with CLI-based tools, such as Azure CLI and Visual Studio code.

Username and You authenticate by specifying a username and password; it is not recommended as passwords are sent over the network.

This is used when you have federated your Windows Server Active Directory with Microsoft Entra ID, allowing Microsoft Entra ID to handle access control. Thus, when the Windows credentials are used, the user is authenticated against the Active Directory, and upon success, the authentication token is taken from Microsoft Entra ID to return it to SQLServer.

Universal with multifactor Here you authenticate using the multifactor authentication. It is mostly used for standard interactive authentication with SQL Server using tools such as SQL Server Management Studio (SSMS).

Service You connect to SQL Server using the client ID and secret of a service principal identity and is mostly created in scenarios when you want to login to SQL Server through applications, jobs, or automated tools.

Managed Special type of service principals and those that work similarly to them. You have two types of managed identities; system-assigned and user-assigned.

Access You authenticate using the access tokens, normally used with non-GUI clients like invoke-sqlcmd.

For details, refer: Microsoft Entra authentication for SQL Server overview - SQL Server | Microsoft Learn

Microsoft Purview Integration:

Microsoft Purview is an Azure based service that helps you to govern, protect and manage data across your organization. With SQL Server 2022, you can now reap the benefits of the Azure based Microsoft purview service on SQL Server 2022 instances on non-Azure environments, by connecting your SQL Servers to Azure through Azure Arc. We discuss more about Azure Arc in the chapter- “Realms of SQL” later in this book.

Using Microsoft Purview for SQL Server 2022, you can extract metadata like instances, databases, schemas, tables with column names and views; you can then classify and label data across databases and instances of SQL Server. You can run the scans across your data estate regularly to ensure you always are looking at the latest classified and labelled data.

For details, refer to Connect to and manage on-premises SQL server instances | Microsoft Learn

Ledger:

If you are looking to bring the power of blockchain to your SQL Server that protects the integrity of the data stored in the database by providing tamper evidence capabilities, even against high-privileged users like database administrators, which ensures you meet your audit requirements, then Ledger is the feature that you should enable for SQL Server 2022.

These are transparent to the application and changes happen only on the database side.

When a database is enabled with the ledger feature, all rows that are been modified by a transaction is represented by a root hash or database digest that is generated using SHA-256 algorithm using a Merkle tree data structure. All the transactions that the databases processes are then also hashed together forming a block. Finally, the block is then SHA-256 hashed through the root has of the current block, and the root hash of the previous block is provided as input, thus forming a blockchain. These database digests can be saved outside of SQL Server on storages such as Azure Block storage configured with immutability policies or Azure confidential ledger. The digests can be used to verify the integrity of the database by comparing the value of the hash in the digest against the calculated hashes in databases to ensure it is tamper-proof.

You can enable the ledger functionality to tables in two forms:

Updateable ledger tables, allowing you to update and delete rows in your tables. It basically enables the system-versioned tables that store the previous version of each row that is updated or deleted in the ledger table. These history tables are automatically created.

Append-only ledger tables, allowing insertions only to the tables. As these are insert only tables, they don’t have the history tables as there is no history to capture.

For more information on this feature, please refer: Ledger overview - SQL Server | Microsoft Learn

Availability

Link to Azure Managed Instance:

If you are looking to modernize your data estate to spread across the cloud and on-premises, then this is one of the features that you can look to enable with SQL Server 2022. When you create a Link from your on-premises SQL Server 2022 to Azure SQL Managed Instance (Azure SQL MI), you can then replicate your data near real-time from SQL Server 2022 to Azure SQL MI. You could use this as a disaster recovery feature and at the same time, your read only queries can be diverted to the Azure SQL MI to balance the load.

The link uses the Distributed Availability Group (DAG), we talk about this feature in detail in the Chapter 5, Business Continuity Strategies for SQL You can also create this Link as a one-way link between previous versions of SQL Server (2016 and 2019) and Azure SQL MI. But, once you perform a manual failover as part of the disaster recovery, you cannot go back to the previous SQL Server primary as that is not supported. But, with SQL Server 2022, you can always failback to and from Azure SQL MI. Basically, the link to Azure SQL MI can be used for migrating to Azure, balancing the workload across the current on-premises and Azure SQL MI by offloading read only workloads to Azure SQL MI, or as a disaster recovery site.

You can learn more about this feature here: Managed Instance link feature overview - Azure SQL Managed Instance | Microsoft Learn

Contained Availability Group:

Based on my experience, working with multiple customers one of the most requested features for availability groups was to move system database objects like logins and jobs to the secondary instance, such that in case of a failover of a database, the logins and jobs are also present on the secondary and does not require manual intervention to sync these up when a failover occurs.

This exact feedback has been addressed in SQL Server 2022, with containers availability group, which gives you the ability to also add relevant portions of the system databases that are master and msdb to be replicated to the other replicas of the availability group, thus when a failover occurs the application can continue working normally as the logins and jobs are already created and present on the secondary replicas.

We talk about the contained availability group in Chapter 5, Business Continuity Strategies for SQL If you like to read more about this feature please refer: What is a contained availability group? - SQL Server Always On | Microsoft Learn

Intelligent Query Processing:

The Intelligent Query Processing (IQP) is a suite of features that are intended to improve the performance of queries and existing workloads with minimal implementation efforts as it depends on Query store. Query Store is a database level feature which when enabled collects and stores query plans, statistics, and other runtime details for some of the queries

running as part of the workload. IQP uses this data collected by the query store to improve the performance of the queries, without manual intervention.

IQP was first introduced in SQL Server 2017, and since then every release has been adding functionalities to improve this suite further. SQL Server 2022 continues this by adding multiple features the most interesting ones to me being:

Query Store on secondary This feature now enables the query store on secondary replicas, and the replicas now send the query execution information to other replicas that previously were available only on the primary replica. In essence, there is one query store shared between primary and secondary replicas. This exists on the primary and stores data for all replicas together.

Parameter Sensitive Plan Have you heard or faced scenarios where a stored procedure that was working fine for days and weeks, suddenly performed bad? Well, this is the known issue of parameter sniffing issues, where a parameterized plan that is cached for a query is not the optimal plan for all the parameter values of that query. This is where this feature comes in and when enabled, it automatically enables multiple, active cached plans for a single parameterized statement, thus ensuring that the right cached plan is picked based on the parameter value.

To learn more about all the latest IQP features, read this article: Intelligent query processing - SQL Server | Microsoft Learn

T-SQL Language enhancements

SQL Server 2022 adds to the existing T-SQL functionalities by expanding the system functions enabling T-SQL developers to do more, without learning a different language. A few examples consist of new functions added, making it simpler to work with JSON data and time-series-based data.

The new functions added to work with JSON in SQL Server 2022 are:

Constructs JSON object text from zero or more expressions.

Constructs JSON array text from zero or more expressions.

Tests whether a specified SQL/JSON path exists in the input JSON sting.

The new Time-Series functions last_value() have been added giving the ability to store and analyze data that changes over time, using time-windowing, aggregation, and filtering capabilities.

To learn about all the latest enhancements in the T-SQL language, please visit the official Microsoft documentation What's new in SQL Server 2022 - SQL Server | Microsoft Learn that summarizes all these features.

Conclusion

To summarize, the concepts that we learned in this chapter, we started with SQL Server installation on Windows and Linux and then connected to the instance using tools like SQL Server Management Studio (SSMS) and Azure Data Studio (ADS). Then we focused our attention on the internals of SQL Server where we learnt about the SQL Server databases and transaction log architecture, different types of system objects including tables and indexes. Finally, we looked at the overall backup & restore strategies and then focused on the latest SQL Server 2022 releases, looking at some of the interesting new features.

In the next chapter, we explore the various SQL offerings and guide you through the decision-making process of choosing the SQL option that best meets your needs.

Additional Resources

SQL Server Management Studio (SSMS), SQL Server Management Studio (SSMS, Microsoft Learn

What is Azure Data Studio - Azure Data Studio, Microsoft Learn,

SQL Server installation guide - SQL Server | Microsoft Learn

Installation guidance for SQL Server on Linux - SQL Server | Microsoft Learn

Pages and Extents Architecture Guide - SQL Server | Microsoft Learn

sys.databases (Transact-SQL) - SQL Server | Microsoft Learn

SQL Server transaction log architecture and management guide - SQL Server | Microsoft Learn

tempdb database - SQL Server | Microsoft Learn

FILESTREAM (SQL Server) - SQL Server | Microsoft Learn

Introducing data virtualization with PolyBase - SQL Server | Microsoft Learn

SQL Graph Architecture - SQL Server | Microsoft Learn

Columnstore indexes: Overview - SQL Server | Microsoft Learn

Introduction to Memory-Optimized Tables - SQL Server | Microsoft Learn (

Ledger overview - SQL Server | Microsoft Learn

Create Clustered Indexes - SQL Server | Microsoft Learn

Create Nonclustered Indexes - SQL Server | Microsoft Learn

Create Unique Indexes - SQL Server | Microsoft Learn

Create filtered indexes - SQL Server | Microsoft Learn

SQL Server and Azure SQL index architecture and design guide - SQL Server | Microsoft Learn

bw-tree-icde2013-camera-ready.dvi (microsoft.com)

https://download.microsoft.com/download/8/3/6/8360731A-A27C-4684-BC88-FC7B5849A133/SQL_Server_2016_In_Memory_OLTP_White_Paper.pdf

Indexes - SQL Server | Microsoft Learn

Recovery Models (SQL Server) - SQL Server | Microsoft Learn

BACKUP (Transact-SQL) - SQL Server | Microsoft Learn

Transaction log backups - SQL Server | Microsoft Learn

Differential Backups (SQL Server) - SQL Server | Microsoft Learn

Complete Database Restores (Full Recovery Model) - SQL Server | Microsoft Learn

File Restores (Full Recovery Model) - SQL Server | Microsoft Learn

Restore Pages (SQL Server) - SQL Server | Microsoft Learn

Piecemeal Restores (SQL Server) - SQL Server | Microsoft Learn

Apply Transaction Log Backups (SQL Server) - SQL Server | Microsoft Learn

What’s new in SQL Server 2022 - SQL Server | Microsoft Learn

Create Azure Synapse Link for SQL Server 2022 - Azure Synapse Analytics | Microsoft Learn

SQL Server containers on Kubernetes with S3-compatible object storage - Getting started - Microsoft Community Hub

Microsoft Entra authentication for SQL Server overview - SQL Server | Microsoft Learn

Connect to and manage on-premises SQL server instances | Microsoft Learn

Managed Instance link feature overview - Azure SQL Managed Instance | Microsoft Learn

What is a contained availability group? - SQL Server Always On | Microsoft Learn

Intelligent query processing - SQL Server | Microsoft Learn

What’s new in SQL Server 2022 - SQL Server | Microsoft Learn

CHAPTER 2

Realms of SQL – Part 1

Introduction

SQL Server has been a successful database in the market for more than two decades. With the advent of Cloud computing, SQL Server continued to evolve and so did the array of deployment options, all designed to accommodate the diverse preferences of our customers and users. Today, we boast of an extensive suite of SQL offerings, spanning Windows, Linux, Containers, Kubernetes, and Cloud deployments.

The topic “Realms of SQL” is divided into two chapters as we need to delve into each of the SQL Ecosystem options and when you should be using those, this chapter focuses on the overall SQL Server Ecosystem and how to arrive at specific choices available in the ecosystem.

Structure

In this chapter the following topics are covered:

SQL Ecosystem

Types of Workloads

Size, Volume and Scalability of the data layer

Data layer choices for Modern cloud-based vs Traditional Applications

SQL Server

Hybrid deployments using Azure Arc

SQL Ecosystem

It is great to employ compelling visualizations at conferences to elucidate the myriad deployment options currently accessible for SQL, allowing attendees to select their preferred platform and environment. A visual presentation like the Figure 2.1 never ceases to captivate and engage the audience.

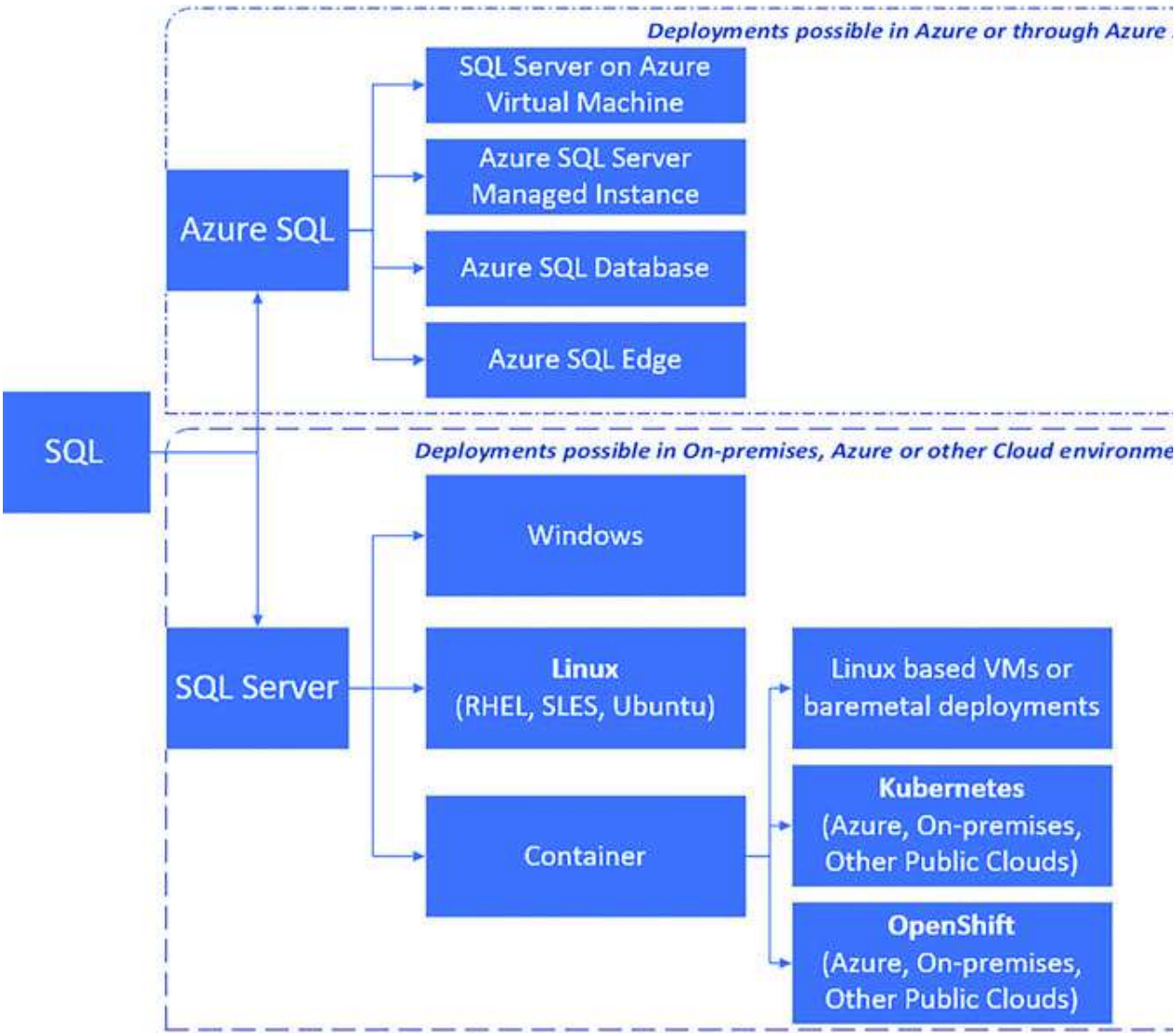


Figure 2.1: SQL Server Deployment Options

A common follow-up question that arises after presenting this Figure 2.1 in various presentations is:

“How do you determine the most suitable data solution for your specific needs?”

When considering your data estate, you might want to reflect on various deployment options. These include Azure SQL or SQL Server on-premises, SQL Server on Windows, Linux, or Containers deployed on Kubernetes, among others.

Structured vs. Unstructured vs. Semi-structured Data

As a database Architect, one of the primary questions that you should ask your application architect team is:

What type of data will the application be storing in the data layer?

Is this going to be Structured, Unstructured, Semi-structured, or a mix of the three?

three?
three? three? three? three?
three? three? three? three?
three? three?

Table 2.1: Types of DataTypes

SQL Server is a relational database management system that not only stores structured data type but is also the perfect platform to store a mix of structured, unstructured, and semi-structured data. Here is the list of features in SQL Server for storing unstructured or semi-structured data:

This feature allows you to store and manage unstructured data, such as documents, images, and videos, directly in the file system while maintaining transactional consistency with the structured data in the database. For more details, refer to the official Microsoft documentation on FILESTREAM.

Introduced in SQL Server 2017, this feature enables you to model and query data in a graph format, which is particularly useful for representing complex relationships and hierarchical data. You can define components like nodes and edges and use T-SQL match functions to find patterns and relationships between different tables.

JSON and XML

SQL Server includes an XML data type, allowing you to store, query, and manipulate XML data using XPath expressions. You can also create indexes on XML data to improve performance.

While SQL Server does not have a native JSON data type, you can store JSON data as NVARCHAR or VARCHAR. T-SQL functions like and others help you parse, query, and manipulate JSON data.

Types of Workload

Once you have identified the data type, the discussion next is about the workload type. With relational databases, normally there are three types of workloads:

OLTP (Online Transaction This type of workload is mostly characterized by a large number of simple transactions comprising insert, update, and delete. These transactions can be write or read-intensive. One of the other common requirements for this type of workload is that it requires multi-user access to the same data which is normally controlled using the Isolation levels.

For example, consider an e-commerce website like Amazon/Flipkart. Every time a customer places an order, updates their cart, or deletes an item, these actions are processed as OLTP transactions. The system needs to handle numerous such transactions simultaneously while ensuring that each user's data remains consistent and isolated from others.

OLAP (Online Analytical Processing): This system is best suited to perform analysis and provides business intelligence by processing large quantities of data that involve aggregations and joins over large data sets. Typical use cases for this type of system are for reporting and business intelligence. These are normally heavy read and aggregation-based workloads.

For example, think of a retail chain like Walmart/DMart analyzing sales data across all its stores. An OLAP system would be used to aggregate sales data, generate reports, and provide insights into trends such as which products are selling the most or which stores are performing best. This helps in making informed business decisions based on comprehensive data analysis.

HTAP (Hybrid Transactional/Analytical Processing): As the name suggests, this is a system that allows both OLTP and OLAP workloads on the same system. They normally remove the requirement or dependency

of moving the data also known as ETL (Extract, Transform and Load) jobs to move the data from the OLTP system to the OLAP system for reporting and analytics. One such example in SQL Server is you can use both Row-based storage for OLTP workloads and a few tables can be converted into a columnar structure with the help of column store indexes (Columnstore indexes: Overview - SQL Server | Microsoft Learn: that support faster analytics, reporting and warehousing queries.

Size, Volume, and Scalability of the Data Layer

The initial two questions are aimed to help guide you in making the optimal database system choice for your requirements. Upon confirming your selection of SQL Server as the chosen database system, the next steps should help you address the deployment option from the fleet of deployment options that you've seen earlier. You first might want to zero down on the deployment being On-premises, Cloud or Hybrid.

During this phase, it's crucial to assess the data's magnitude, taking into account not only its present scale but also conducting essential discussions with the application team. Additionally, perform trend analysis by examining similar environments deployed in the past to pinpoint growth rates expected over the next several years. This comprehensive approach ensures the construction of a scalable environment that aligns with both current application demands and future needs.

It's imperative to gain insights into the anticipated data generation rate by the application for processing and to align this with the business's expectations regarding data storage and retrieval. Concurrently, it's essential to engage in discussions about the required Recovery Point Objective (RPO) and Recovery Time Objective (RTO) in the event of a database crash or failure. This holistic assessment forms the basis for making an informed decision about the appropriate hosting location for your SQL Server.

As a general recommendation, if your key requirements include the ability to scale resources, such as storage, compute, and memory independently, along with high data resilience across multiple geographical locations, cloud-based deployment offerings are typically preferred. This approach offers flexibility, scalability, and robust data redundancy without requiring extensive initial investments in physical infrastructure. In this chapter, we will look at Microsoft's Azure SQL offerings and how you can get started with them.

If your organization already possesses interconnected multi-region data centers and a proficient team capable of establishing and maintaining high resilience across these data centers, then choosing an on-premises deployment can also be a viable option. In this scenario as well, you have several SQL offerings to select from when deploying on-premises, including SQL Server on Windows, Linux, or containers. This option provides you with flexibility and control over your database infrastructure while leveraging your existing data center resources and expertise.

Modern, Newborn Cloud Application vs. Migration of an Existing Application

This is another essential point to consider and discuss with your application team, as a thorough understanding of this tenet will help you determine the optimal SQL Offering for your application workload.

If you are developing a new cloud-native application with a focus on multi-region availability and functionality, then a fully managed database service like Azure SQL Database may be the best fit. However, if you are dealing with an existing application originally designed for on-premises deployment and are currently in the process of modernizing it, a Hybrid Cloud offering or Azure SQL Server Virtual Machines might be a better fit. In this case, a “lift and shift” type of migration may be more suitable than transitioning to a fully managed database service. This decision depends on the specific requirements and nature of your application.

Budget and Skillset Considerations

Lastly, one more point to focus on is Budget and skillset considerations.

When we discuss budget, it's essential to take into account both Capital Expenditure (CapEx) and Operational Expenditure (OpEx). Small and medium-sized businesses often aim to initiate their operations with minimal upfront expenses, avoiding the costs of establishing and maintaining data centers and hiring and retaining skilled professionals for database system configuration and maintenance. In such cases, cloud-based deployments typically offer the best fit.

Conversely, large enterprises that have already established multi-region data centers and have a robust on-premises infrastructure may find on-premises deployments to be a better fit. These enterprises often seek complete control and administration of their database environments and are willing to invest in the necessary resources to maintain their infrastructure.

The choice between cloud-based and on-premises deployments should align with your organization's budgetary constraints and the availability of the required skillsets to manage and operate the chosen database solution effectively.

When we talk about budget, we need to consider both CapEx and Opex. Small and medium businesses tend to focus on getting started with

minimum expenditure on datacenters, hiring and maintaining skilled professionals to configuring and maintaining database systems, thus for them Cloud-based deployments are the best fit. At the same time, for large enterprises that already have multi-region data centers set up and working, on-premises deployments might be a better fit as they look for complete control and administration of the environment.

SQL Server (on-premises)

Before we investigate and learn about the Azure SQL options and how to get started with them, let's first consider the on-premfc deployment options with SQL Server 2022. When writing this book, SQL Server 2022 is the latest version available for SQL Server and thus, we will be focusing on SQL Server 2022 and not the previous versions.

SQL Server 2022 is supported across Windows, Linux, and Container platforms, such as Kubernetes and Red Hat OpenShift. Though we will look at SQL Server on Windows and how to get started in this chapter, the focus of this section and chapter will be on SQL Server on Linux and containers and Azure-Arc.

SQL Server 2022 on Windows

SQL Server 2022 is compatible with Windows 10, making it ideal for client-based operating systems, and with Windows Server 2016 and later versions, making it suitable for server-based operating systems. To read more about the hardware and software requirements, please refer to the official documentation SQL Server 2022: Hardware & software requirements - SQL Server | Microsoft Learn which is regularly updated.

When we look at the installation experience and options for SQL Server on Windows, you have the traditional wizard-based, command prompt, and the Sysprep (System Preparation) methods of installation. Generally, if you are installing SQL Server on a single new machine or creating a failover clustered instance, you normally will go ahead and run the wizard-based installation. You can invoke the installation of SQL Server using the setup.exe located within the SQL Server setup media; this is termed a command prompt-based installation. There are multiple setup parameters that you can use with the setup.exe command to install SQL Server based on your requirements. If you prefer automated installation, then you choose the command prompt-based installation.

Now you may ask, what is Sysprep used for? So, Sysprep or system preparation is an installation option normally used, when you want to install SQL Server on multiple target systems, where you can have more than one instance on each of the target systems.

Sysprep-based installation is run in two steps:

Prepare Image

Complete Image

In the prepare Image stage, you just lay and install the binaries, without configuring SQL Server, meaning you don’t set the startup accounts for any of the SQL services like the SQL Server or SQL agent which you normally would have configured when doing the installation in any other way. So, you prepare the machine by installing the binaries and at a later point in time you can complete the installation.

In the Complete image stage, you go ahead and configure the SQL Server, such as providing the SQL service startup accounts, computer, and network-specific settings.

The benefit of breaking the setup into two phases of installation and configuration gives you the ability to prepare multiple systems with the basic SQL Server binaries installed, and then allow different users to configure SQL Server according to their requirements. For example, imagine your company has multiple projects and for every project, the need is that they need SQL Server 2022 installed on the Windows Server 2022 operating system, but the startup accounts, the computer and network-specific settings are different for each project; in these scenarios, you have two options to configure SQL Server for each of those projects.

Run SQL Server 2022 installation on each of the machines for every project using the installation wizard or automate all this using the setup.exe, which again requires you to create the script.

Create an image of the Windows Server 2022 (using Windows Sysprep) with the unconfigured but prepared instances of SQL Server using the (SQL Server Sysprep) and then deploy this image across multiple systems and ask each project owner to run the complete configuration for Windows Server 2022 followed by completing the configuration for SQL Server that can be specific to each instance on each system that is being run.

Hopefully, this example helps you understand the use case for the Sysprep option. To read more about this, refer to Microsoft official documentation: Considerations for installing SQL Server using SysPrep - SQL Server | Microsoft Learn

Let's talk about a fantastic new experience that you will notice when you install SQL Server 2022. If you opt to use the normal wizard-based installation, you will immediately notice the first screen that provides you with an additional choice in licensing, that is, "pay-as-you-go" also written as payg licensing for SQL Server on-premises deployment when connected with the Azure shown in Figure

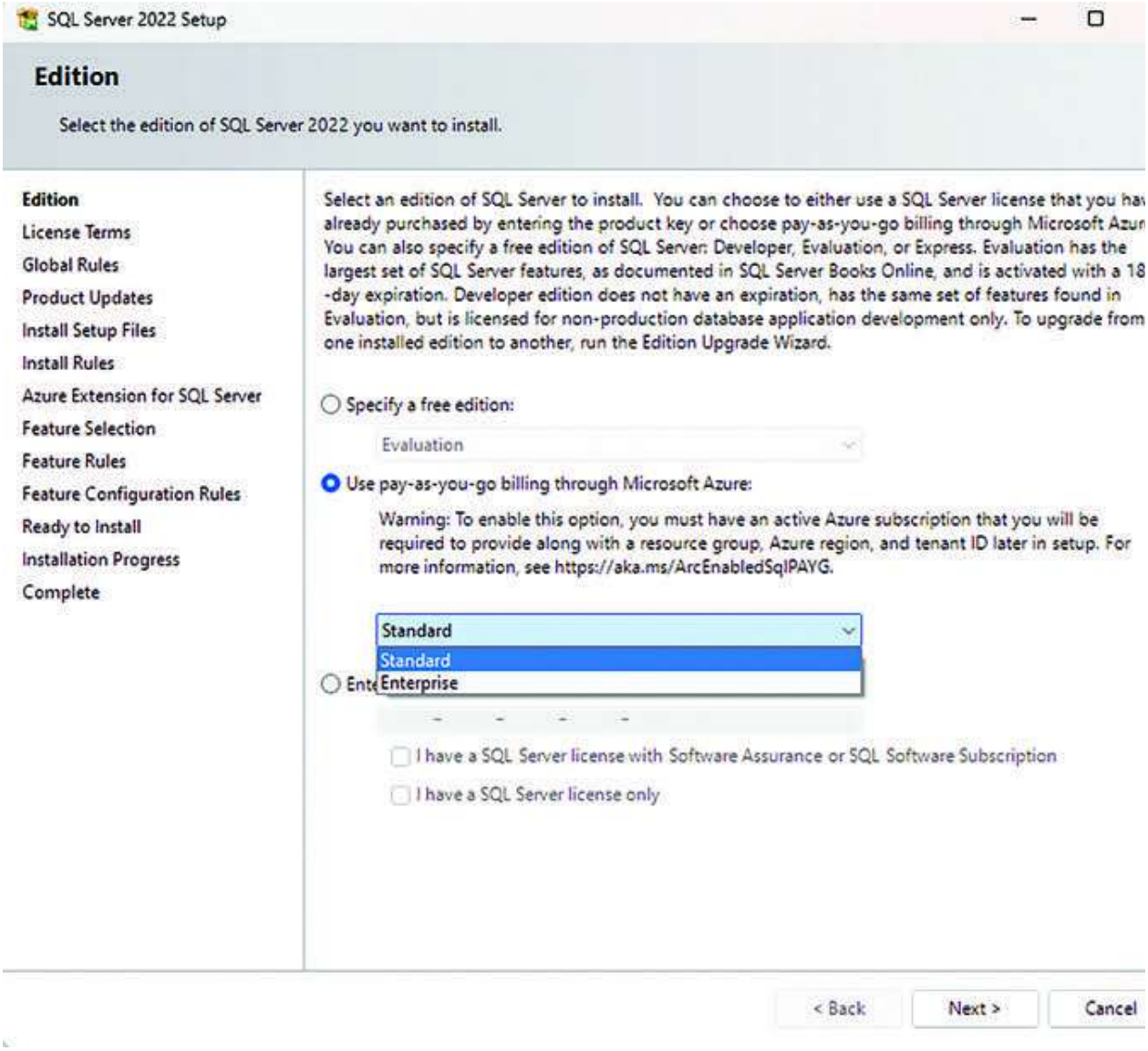


Figure 2.2: New PAYG option with SQL Server 2022 setup

When you choose this option, you get a choice of deploying Enterprise or Standard edition of SQL Server under the PAYG(pay-as-you-go) model. This provides you with benefits like:

Consumption-based licensing, which is typical to cloud deployments but now can be used for on-prem deployments as well.

It also helps you improve licensing cost efficiency by eliminating the requirement to pay full upfront licenses for on-prem deployments and only

pay for what you consume.

Consistent billing for your complete SQL Server deployments across Azure, non-Azure clouds, and On-prem deployments

The hourly charges for SQL Server are based on the edition of SQL Server and the maximum size of the hosting server at any given time. The pricing table is available on the official Microsoft pricing documentation: SQL Server 2022—Pricing | Microsoft For quick reference, the pricing is shown in Table The pricing shown here is as of the time of writing this book. Please note that the pricing may change over time.

time. time.
time.
time.

Table 2.2: SQL Server pricing table reference

The ability to use pay-as-you-go (PAYG) licensing for SQL Server on-premises or anywhere is made possible by Azure Arc-enabled SQL Server agent. The SQL Server 2022 installation wizard also provides an additional screen that allows the installation of the Azure Extension for SQL This extension enables you to connect the SQL Server instance to Azure via Azure Arc-enabled SQL Server. The screen as shown in Figure asks you for the Azure Login details, the Subscription ID, tenant ID of the Azure Subscription, including the resource group name and the region where you want these resources to be added and shown under the Azure Arc blade in the Azure portal. If you don't have an Azure subscription, then you cannot use the Pay As you Go model of billing.

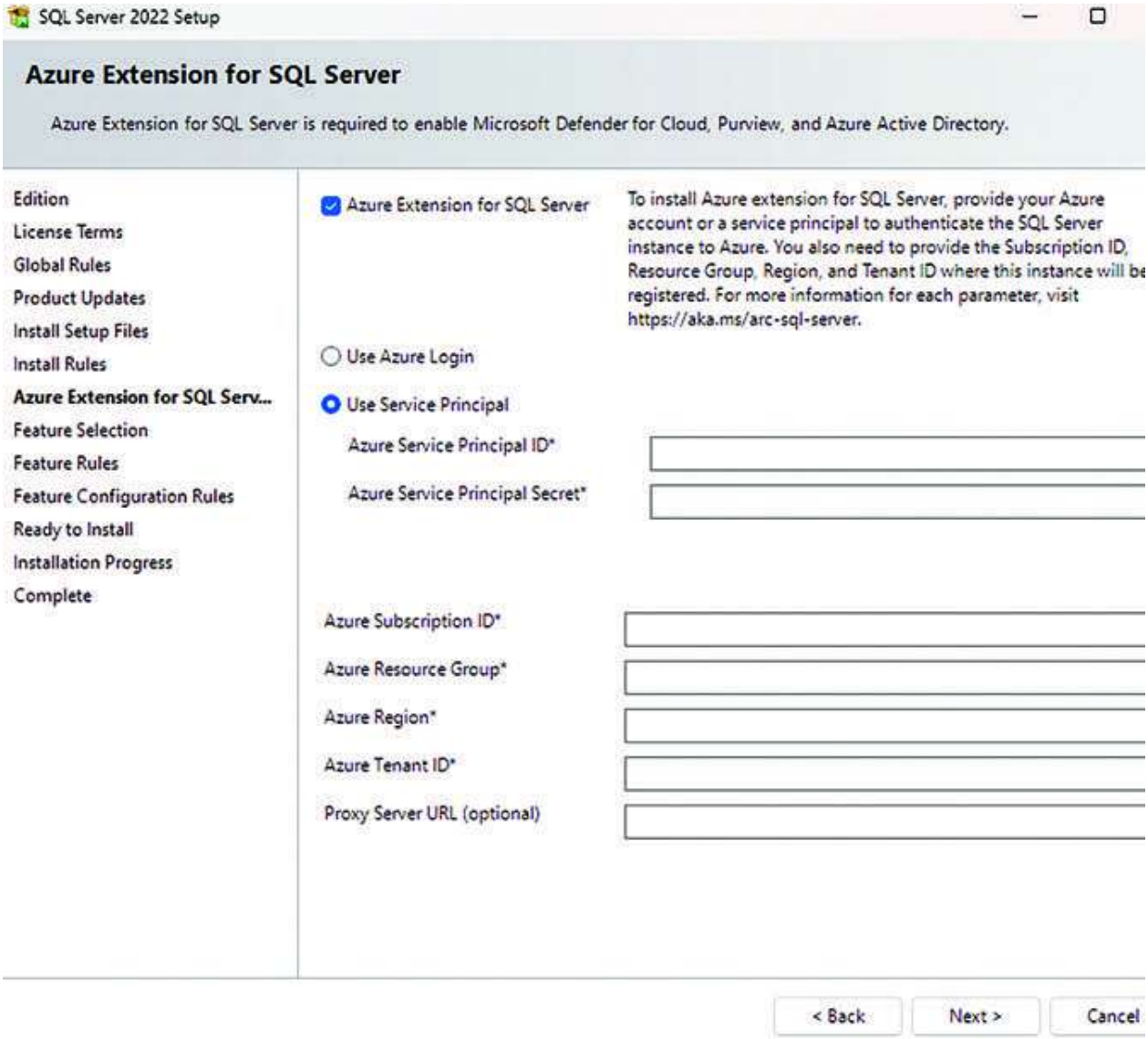


Figure 2.3: Azure Extension installation with SQL Server 2022 setup

This gives a good segue to introducing Azure Arc and how it can help you create a true hybrid environment.

Azure Arc-enabled SQL Server- The Hybrid Enabler

Today, a lot many enterprises have their IT infrastructure diversified, with a few on-premises deployments and a few other servers across multiple Cloud environments, that could be a mix of one or more Cloud providers like Microsoft Azure. Now, with this kind of complex deployment, it has become very hard for companies to have a centralized and unified way to manage and govern their infrastructure. This is where Azure Arc comes in, it enables a centralized and unified way to:

Manage all your resources like servers, Kubernetes Clusters and more deployed across non-Azure and On-premises environments, all managed under a single pane of glass through the Azure Resource Manager.

Azure Arc also enables you to run some of the Azure-specific services like the Azure data services that include SQL-managed instances or Azure PostgreSQL outside of Azure. We will see how this is possible when we talk about Azure Arc-enabled SQL-managed instances later in this chapter.

To enable a resource like a Virtual machine that is deployed on non-Azure or On-prem to be managed and governed via the Azure Arc, you need to install the Azure-connected machine agent. This agent runs on your on-prem or non-Azure machine and connects to Microsoft Azure. As you can imagine, this agent runs as a service on your machine. It consists of multiple logical components as shown in Figure 2.4 which is available and documented on Microsoft's official documentation page: Overview of the Azure Connected Machine agent - Azure Arc | Microsoft Learn

Hybrid Instance Metadata service (HIMDS) that manages the connection to Azure and the connected machine's Azure identity.

The Guest configuration agent helps assess if the machine complies with the required policies to help enforce compliance.

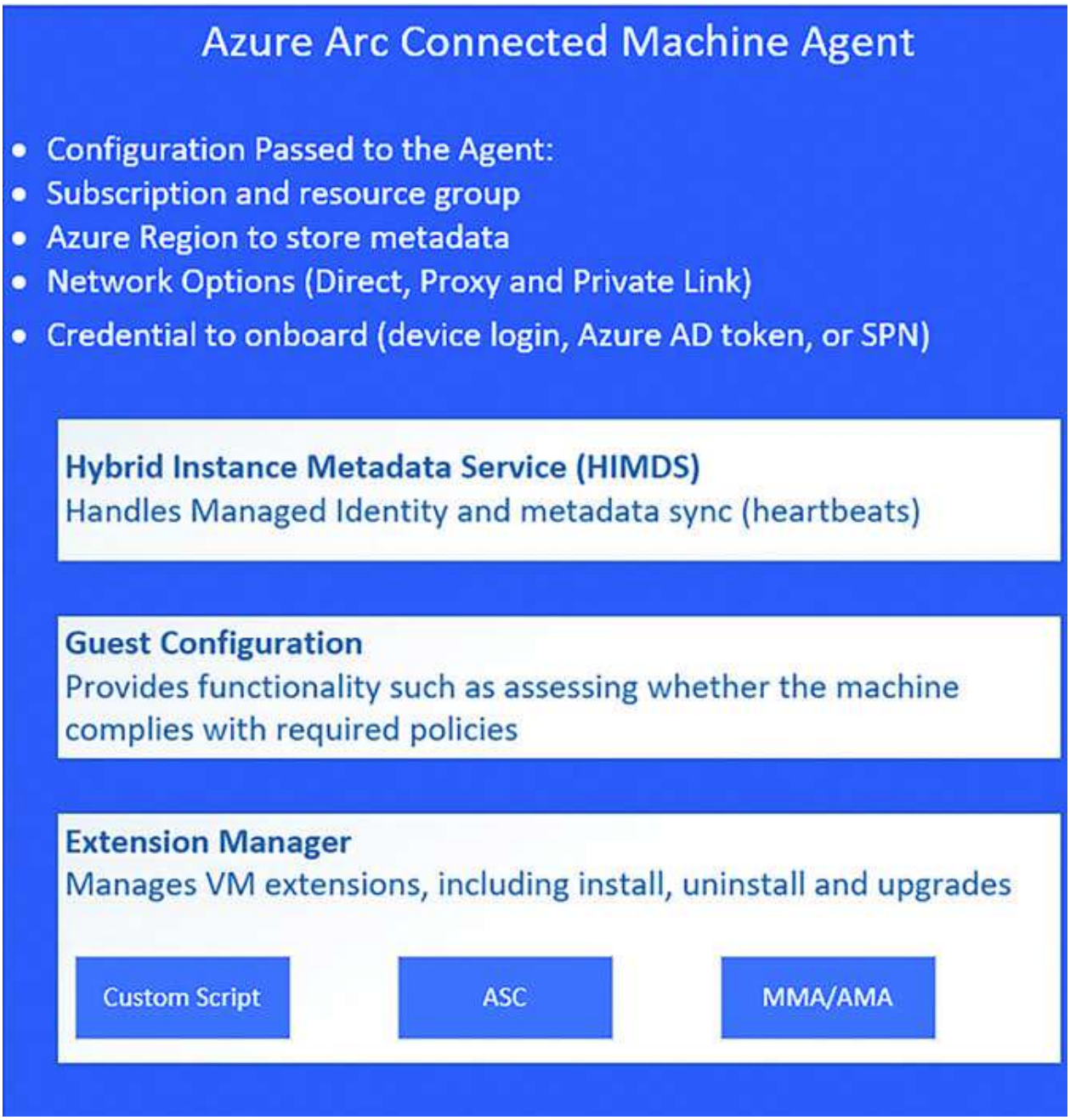


Figure 2.4: Azure Connected Machine Agent components

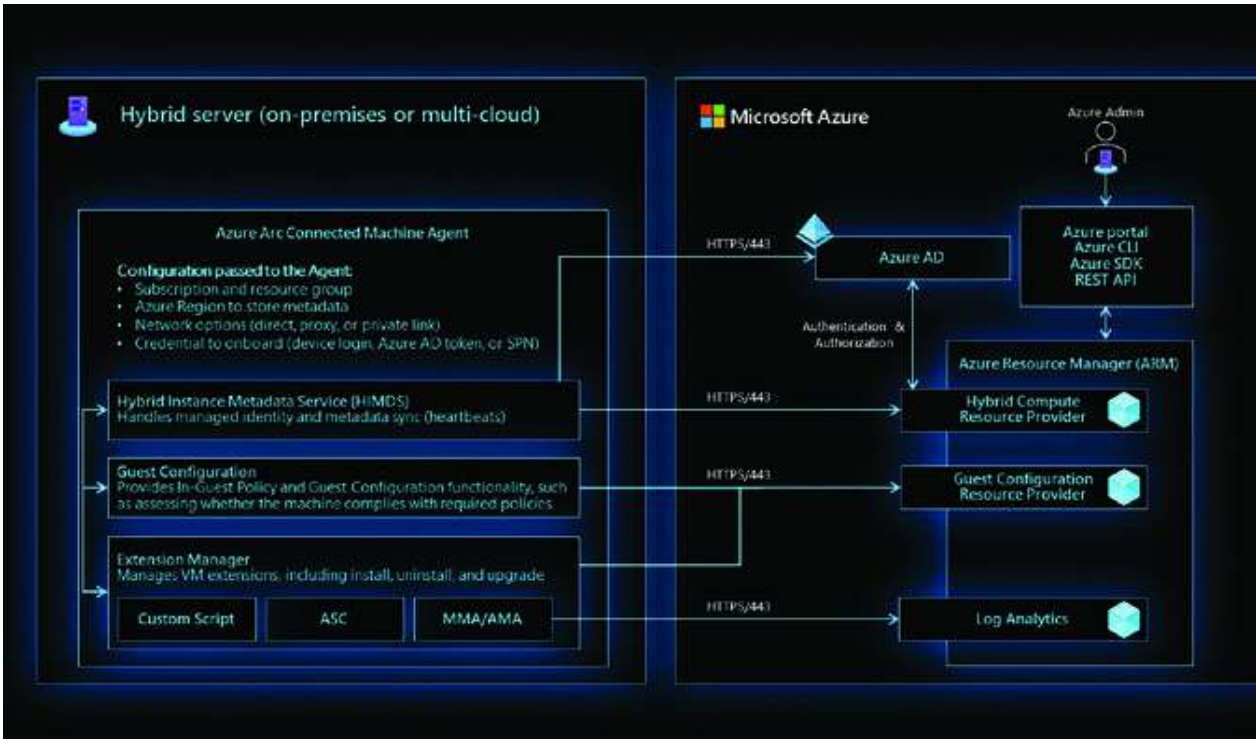


Figure 2.5: Azure Connected Machine Agent components

Let’s look at a quick demo on how easy it is to get started with Azure Arc. The following Figure 2.6 is a snapshot of what the Azure Arc portal looks like.

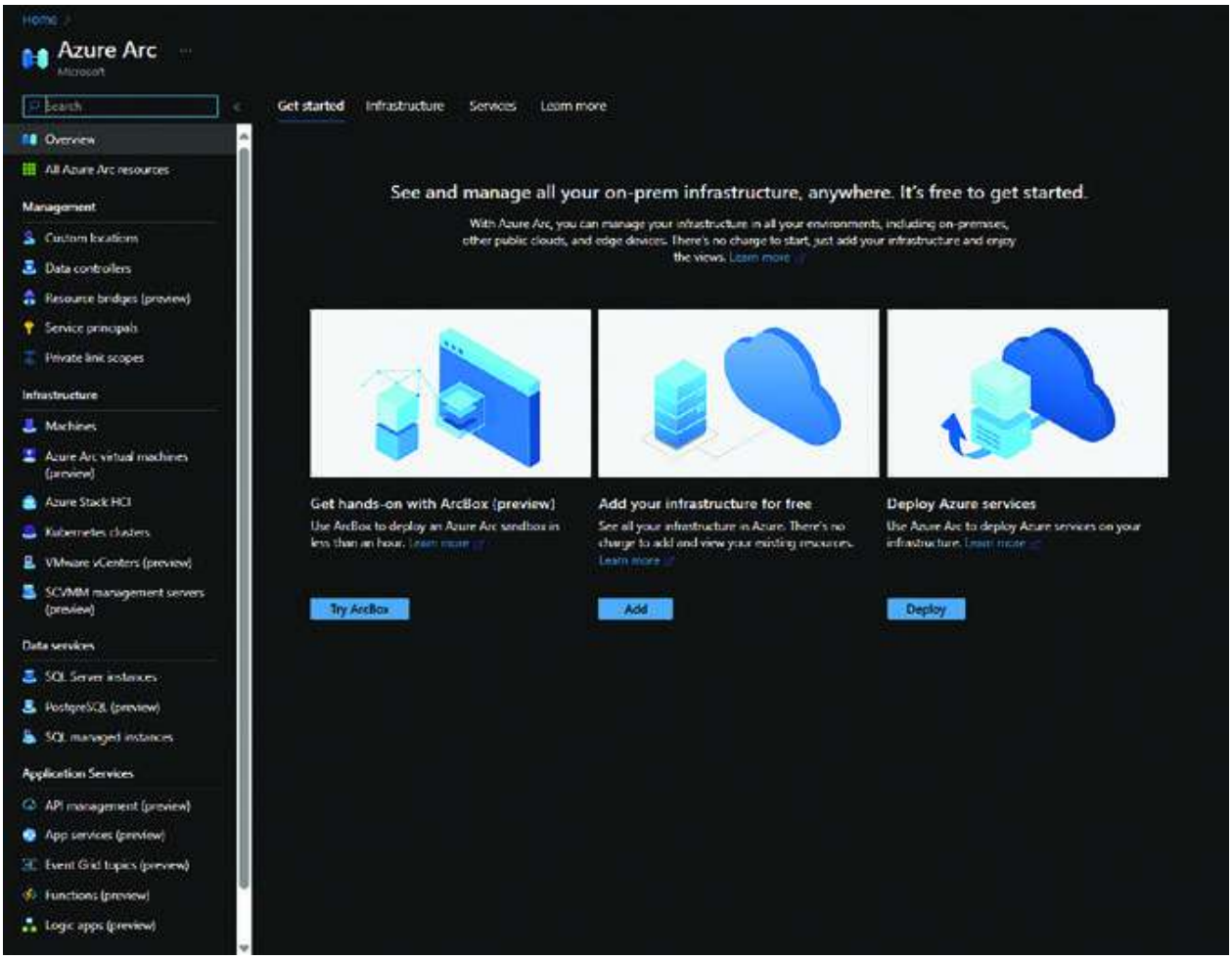


Figure 2.6: Azure Arc start page in the portal

Note, the different Infrastructures, such as the Machines, Kubernetes clusters, Azure Stack HCI, and more that can be on-boarded to Azure Arc, meaning these are all non-Azure resources running anywhere from on-prem to non-Azure clouds that are now connected to Azure and can be managed via Azure.

Azure arc-enabled Servers

Now, let see how easy it is to connect and enlist one of our resources to Azure Arc. We are going to add one of the virtual machines called linuxtestVM which is running Ubuntu 20.04 LTS, that is in Central India.

We will go to the Azure Arc portal and click on the “add server” option as shown in Figure

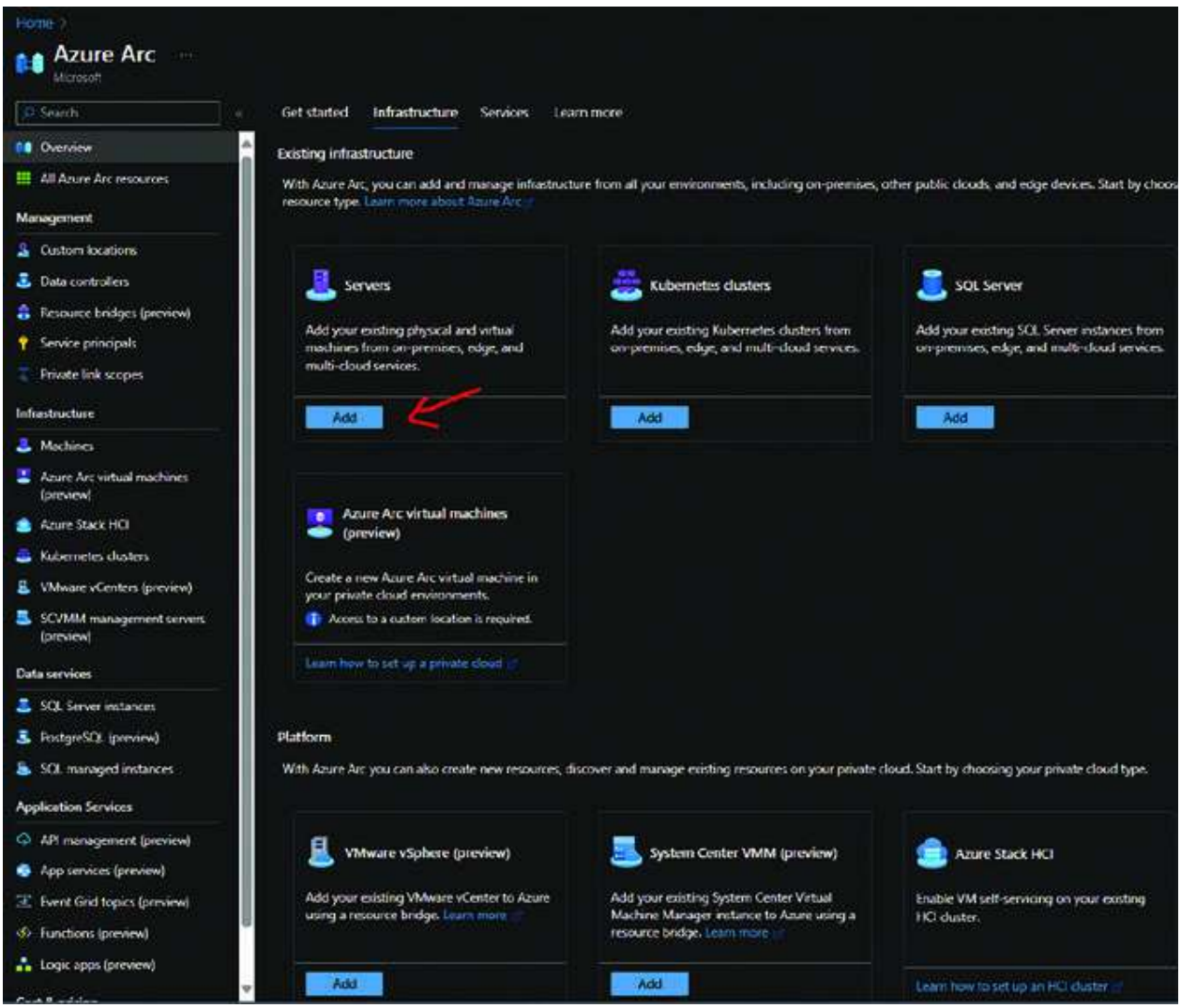


Figure 2.7: Azure Arc portal to add servers

Then, click “add a single server” as we add only one server for this demo; refer Figure 2.8 for details.

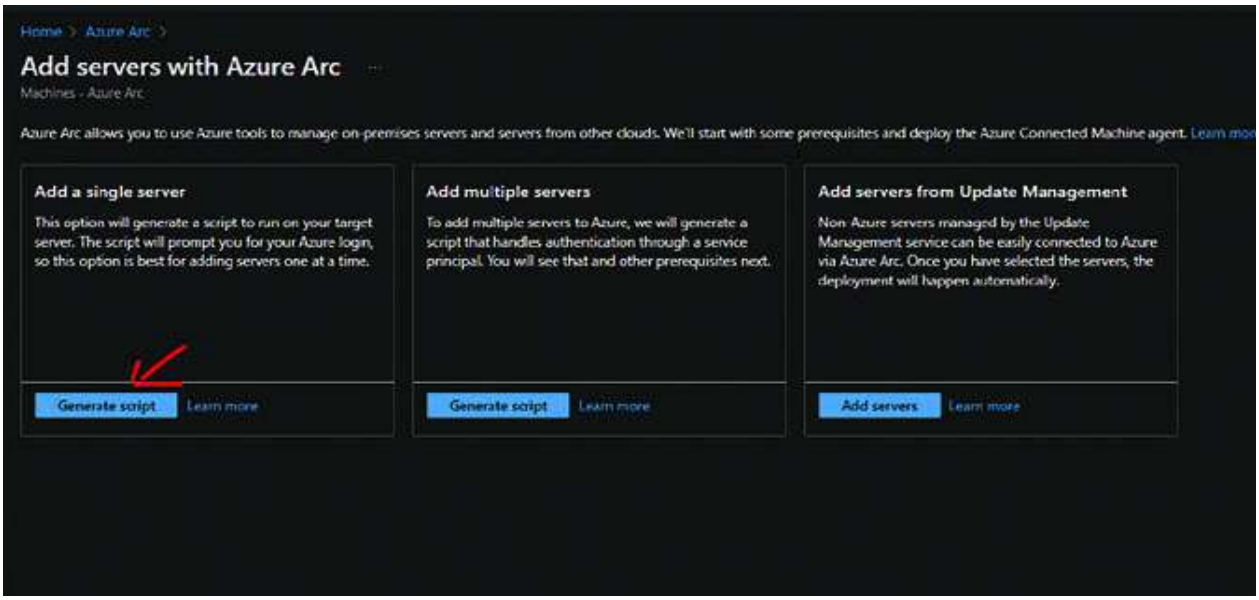


Figure 2.8: Azure Arc portal, options to add servers

This will take me to the script generation section, where we need to provide details such as the resource group name where we want the resource to be listed, the operating system, and the region details, as shown in Figure

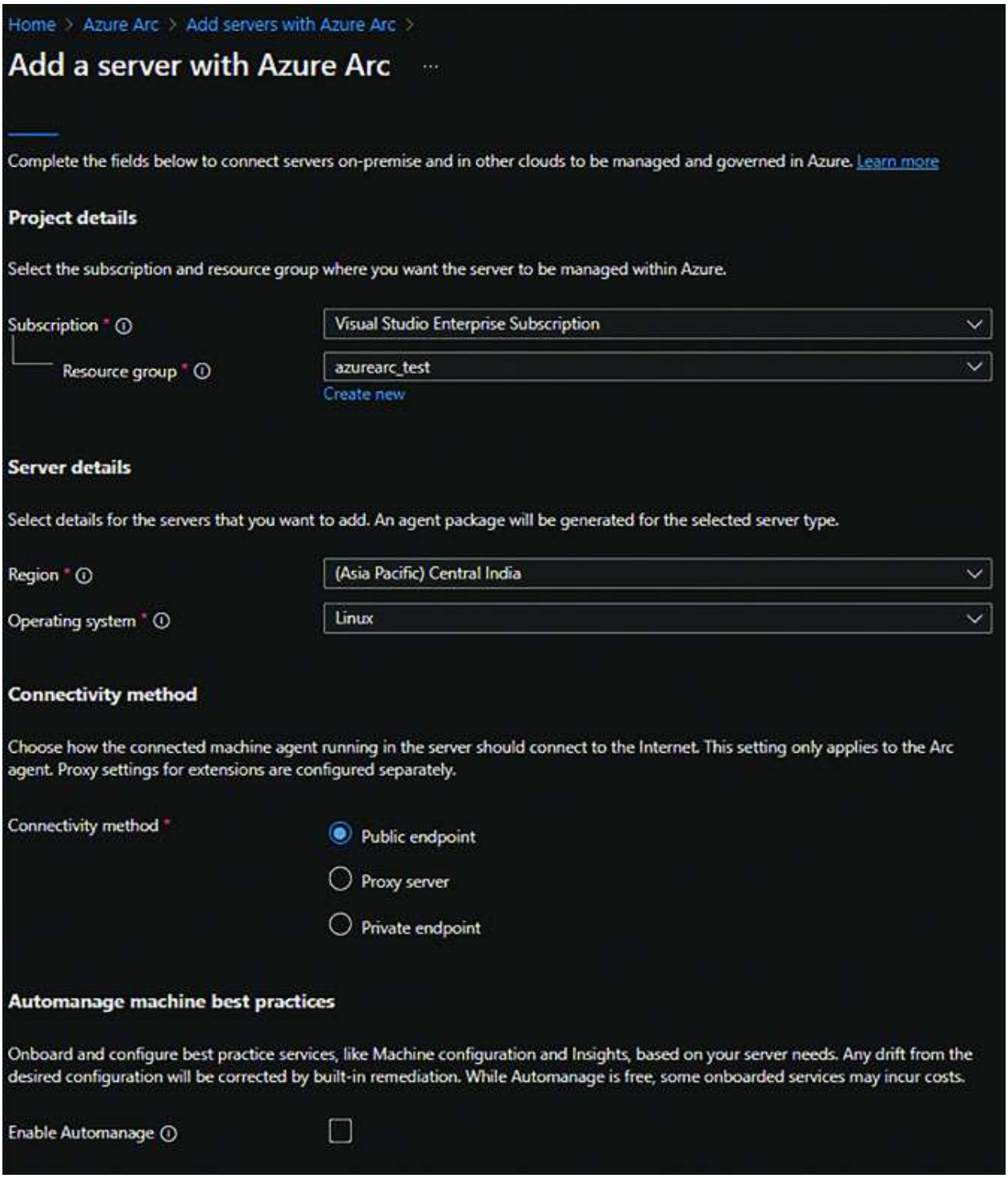


Figure 2.9: Azure Arc portal, server details required

Once, the required details are provided, download the script called the "OnboardingScipt.sh" and run it on the server to be connected and managed using Azure Arc. Let's see what the bash script has; if Linux is used as the

operating system, then it would provide the bash script (.sh); if Windows, then it would provide a PowerShell script (.ps1).

```OnboardingScript
export subscriptionId=”5b65fxxx-xxxx-xxxx-xxxx- xxxxxxxxxx58”;
export resourceGroup=”azurearc_test”;

export tenantId=”9137xxxx-xxxx-xxxx-xxx-xxxxxxxxxx7c”;
export location=”centralindia”;
export authType=”token”;
export correlationId=”2fabxxxx - xxxx - xxxx - xxxx - xxxxxxxxba38”;
export cloud=”AzureCloud”;

# Download the installation package
output=$(wget https://aka.ms/azcmagent -O ~/install_linux_azcmagent.sh 2>&1);
if [ $? != 0 ]; then wget -qO- --method=PUT --body-data=”{\”subscriptionId\”:\”$subscriptionId\”,\”resourceGroup\”:\”$resourceGroup\”,\”tenantId\”:\”$tenantId\”,\”location\”:\”$location\”,\”correlationId\”:\”$correlationId\”,\”authType\”:\”$authType\”,\”operation\”:\”onboarding\”,\”messageType\”:\”DownloadScriptFailed\”,\”message\”:\”$output\”}” “https://gbl.his.arc.azure.com/log” &> /dev/null || true; fi;
echo “$output”;

# Install the hybrid agent
bash ~/install_linux_azcmagent.sh;

# Run connect command
sudo azcmagent connect --resource-group “$resourceGroup” --tenant-id “$tenantId” --location “$location” --subscription-id “$subscriptionId” --cloud “$cloud” --automanage-profile
```

“/providers/Microsoft.Automanage/bestPractices/AzureBestPracticesDevTest ” --correlation-id “$correlationId”;
```

Here, both the subscriptionId and the tenantId are obfuscated for security reasons, but as you can see in the script, it sets details like subscription ID and tenant ID based on the Azure account we used where we wanted the on-prem or non-Azure resources to be managed using Microsoft Azure.

The other details, such as the location and resource group name, will be based on the information provided in the previous step. The machine will authenticate using a one-time token. The machine is going to authenticate via a one-time token.

Then the script goes ahead and downloads the script AzureConnectedMachinAgent which helps install the agent. Once the agent is installed, we run the azcmagent connect command to connect the current server to Azure Arc, we provide the details, such as subscription ID, tenant ID, resource group, and other details.

When we run this on the on-prem machine, this is how it looks.

amvin@linuxtestvm:~$ sudo azcmagent connect --resource-group “azurearc_test” --tenant-id “9137xxxx-xxxx-xxxx-xxx-xxxxxxxxxx7c “ --location “centralindia” --subscription-id “5b65fxxx-xxxx-xxxx-xxxx-xxxxxxxxxx58” --cloud “AzureCloud” --automanage-profile “/providers/Microsoft.Automanage/bestPractices/AzureBestPracticesDevTest ” --correlation-id “2fabxxxx - xxxx - xxxx - xxxx - xxxxxxxxba38”
INFO Connecting machine to Azure… This might take a few minutes.
INFO Testing connectivity to endpoints that are needed to connect to Azure… This might take a few minutes.
```

To sign in, use a web browser to open the page https://microsoft.com/devicelogin and enter the code NHE2VD2BW to authenticate.

20% [==>]
30% [===>]
INFO Creating resource in Azure… Correlation ID=2fabxxxx - xxxx - xxxx - xxxx - xxxxxxxxba38 Resource ID=/subscriptions/5b65fxxx-xxxx-xxxx-xxxx-xxxxxxxxxx58/resourceGroups/azurearc_test/providers/Microsoft.HybridCompute/machines/linuxtestvm
60% [=========>]
80% [============>]
..
…
…
INFO Connected machine to Azure
INFO Machine overview page: https://portal.azure.com/#@9137……..

Once this is done, you can now see on-prem VM in the Azure portal through the Azure Arc blade as shown in Figure

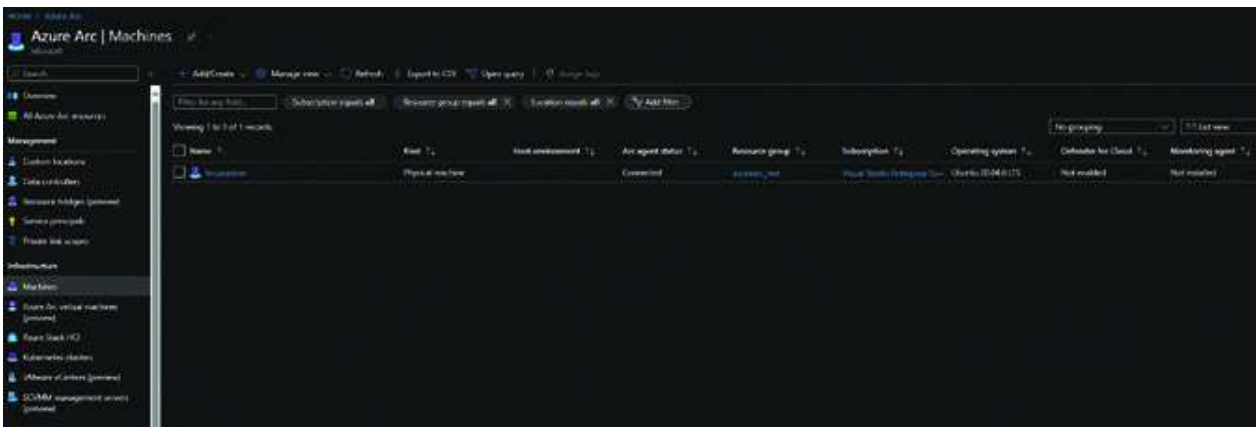


Figure 2.10: Azure Arc portal, showing Azure Arc enabled servers

Further, you can see the essential details of the server like OS, model, and manufacturer details as shown in Figure

Figure 2.11: Azure Arc portal, showing server details.

Azure Arc provides you with the following control plane functionality for no extra cost, for details, please refer to the official Microsoft documentation at Azure Arc overview - Azure Arc | Microsoft Learn

Resource organization through Azure management groups and tags

Searching and indexing through Azure Resource Graph

Access and security through Azure Role-based access control (RBAC)

Environments and automation through templates and extensions

Update management

For details on how you should plan and set up the environment, please refer to the official Microsoft documentation: Plan and deploy Azure Arc-enabled servers - Azure Arc | Microsoft Learn

Azure arc-enabled SQL Server

Now, let us understand Azure Arc-enabled SQL Server. As the name suggests, Azure Arc-enabled SQL Server extends Azure services to SQL Server instances hosted on-premises and/or non-Azure cloud environments, thus enabling centralized SQL Server management. It also helps enable some of the other features, such as:

Pay-as-you-go for SQL Server

Microsoft Defender for Cloud

Extended Security updates

Microsoft Entra ID authentication (previously known as Azure Active Directory (AAD) authentication)

Best Practice assessment

Automated backups and patching

You can have SQL Server 2012 and higher versions running on Windows or Linux (SQL Server 2017 and higher) on-premises or non-Azure cloud, that you can connect to Azure via the Azure Arc-enabled SQL Server and be able to take advantage of these new features, for example, the pay-as-you-go feature is available for all the SQL Server 2012 and higher versions, and also available for SQL on Linux. Thus, Azure Arc-enabled SQL Server gives you the ability to use these new features with older versions of SQL Server.

Note that, while writing this book some of the Azure Arc-enabled SQL Server features, such as Microsoft Defender for cloud, Best Practice assessment, Automated backup and patching do not work for SQL Server installed on Linux, while it does work for SQL Server on Windows deployments. Also, some of these features are not available on all the previous versions of SQL Server. Thus, it is recommended to refer to the official documentation Azure Arc-enabled SQL Server - SQL Server | Microsoft Learn from Microsoft to ensure that you are aware of the latest supported features across different SQL Server versions and Operating systems.

Figure 2.12 is the diagram for the Azure Arc-enabled SQL Server architecture as documented here: Azure Arc-enabled SQL Server - SQL Server | Microsoft Learn

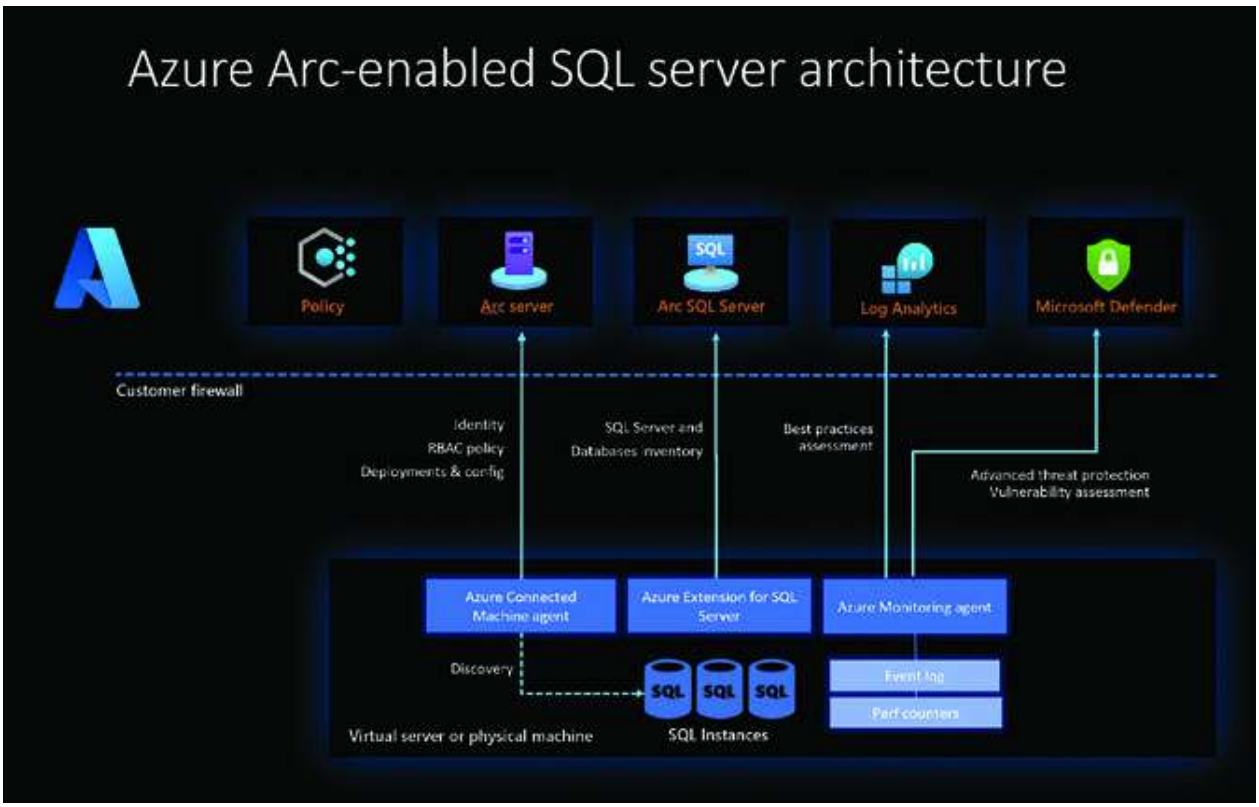


Figure 2.12: Azure Arc-enabled SQL Server architecture

You can see that the virtual server or the physical machine has three agents that are:

Azure Connected Machine Connects the machine to the Arc server

Azure Extension for SQL Enrolls the SQL Server as an Azure Arc-enabled SQL server.

Azure Monitoring Agent Collects the monitoring data from the guest operating system and delivers it to Azure Monitor for use by features, insights and other services like the Microsoft Defender for the cloud.

Let us now enrol one of the local SQL Servers running on Ubuntu to Azure Arc infrastructure. For this demo, the virtual machine is named: linuxvmtest' and has an SQL Server instance installed as shown in Figure

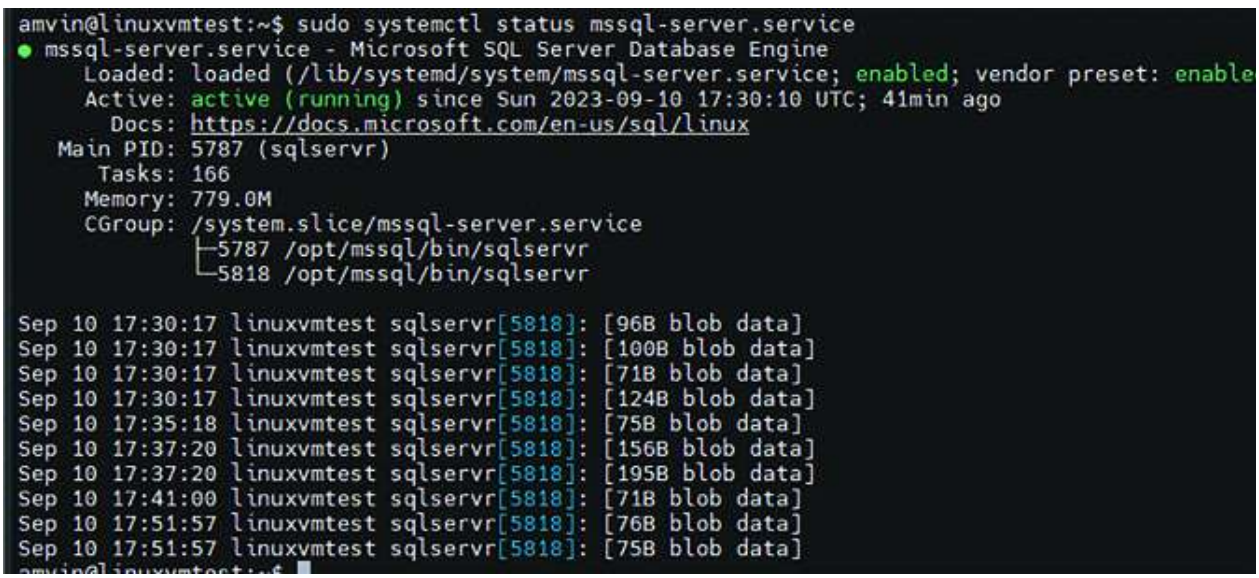


Figure 2.13: SQL Server instance running on VM

We already have the linuxvmtest machine enrolled and have the Azure-connected machine agent running and connected as shown in Figure

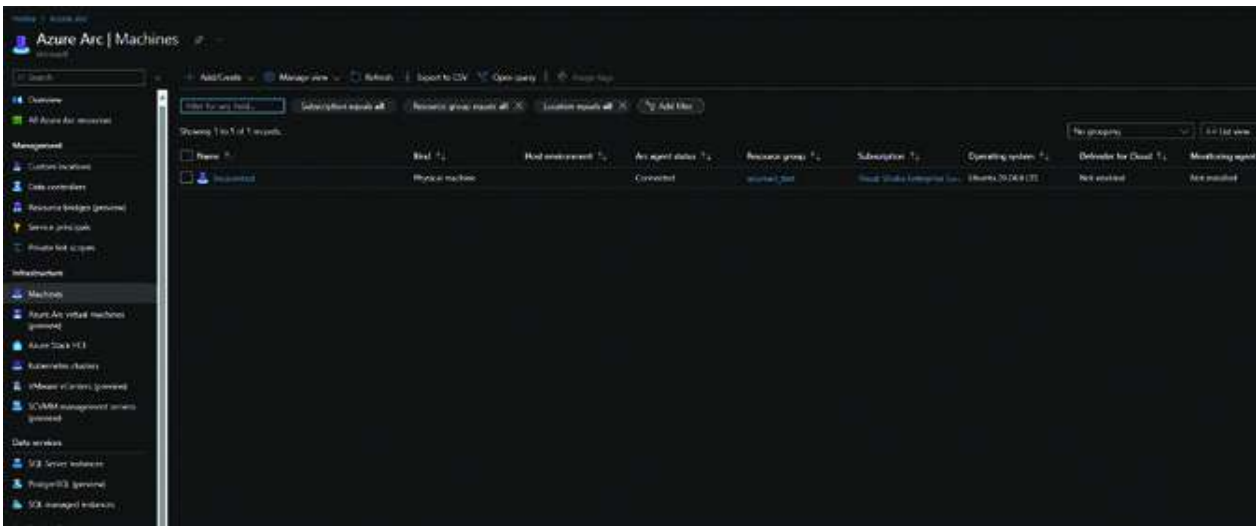


Figure 2.14: Linuxvmtest enrolled as Azure Arc enabled Server

Under the data services, we click on the SQL Server instances and then the ‘add’ button, which then brings up the screen, depicted in Figure you select the connect server option to connect the SQL Server to Azure Arc.

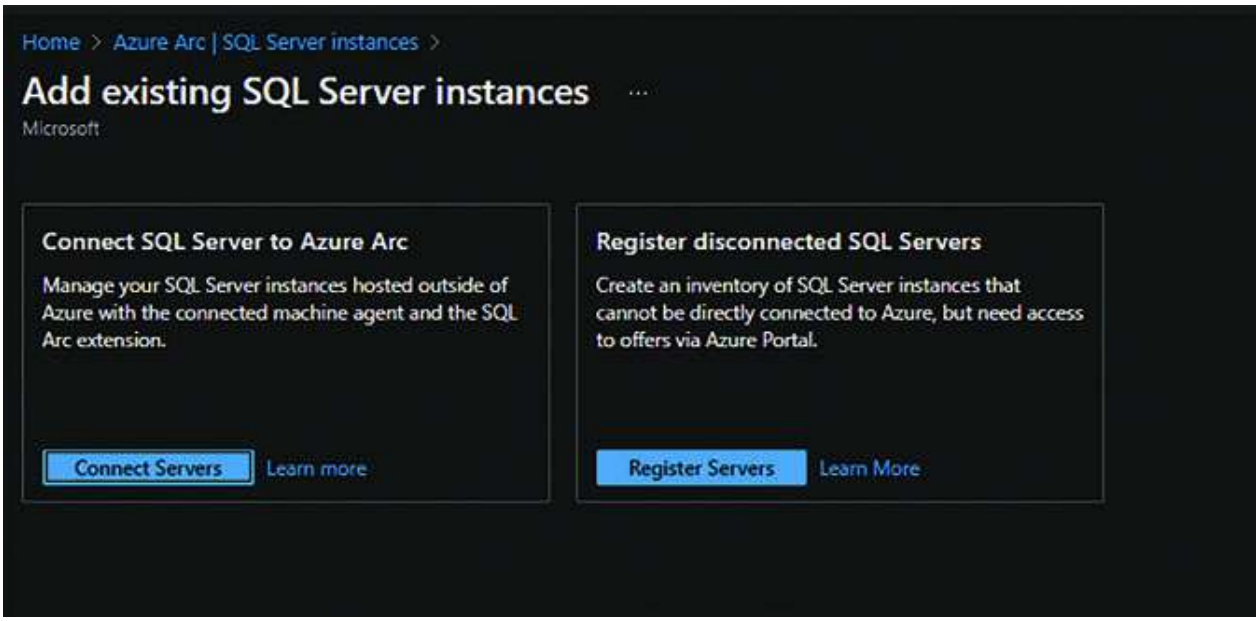


Figure 2.15: Azure Arc portal showing options to add existing SQL Server instances

This brings the following screen shown in Figure and you click on Next.

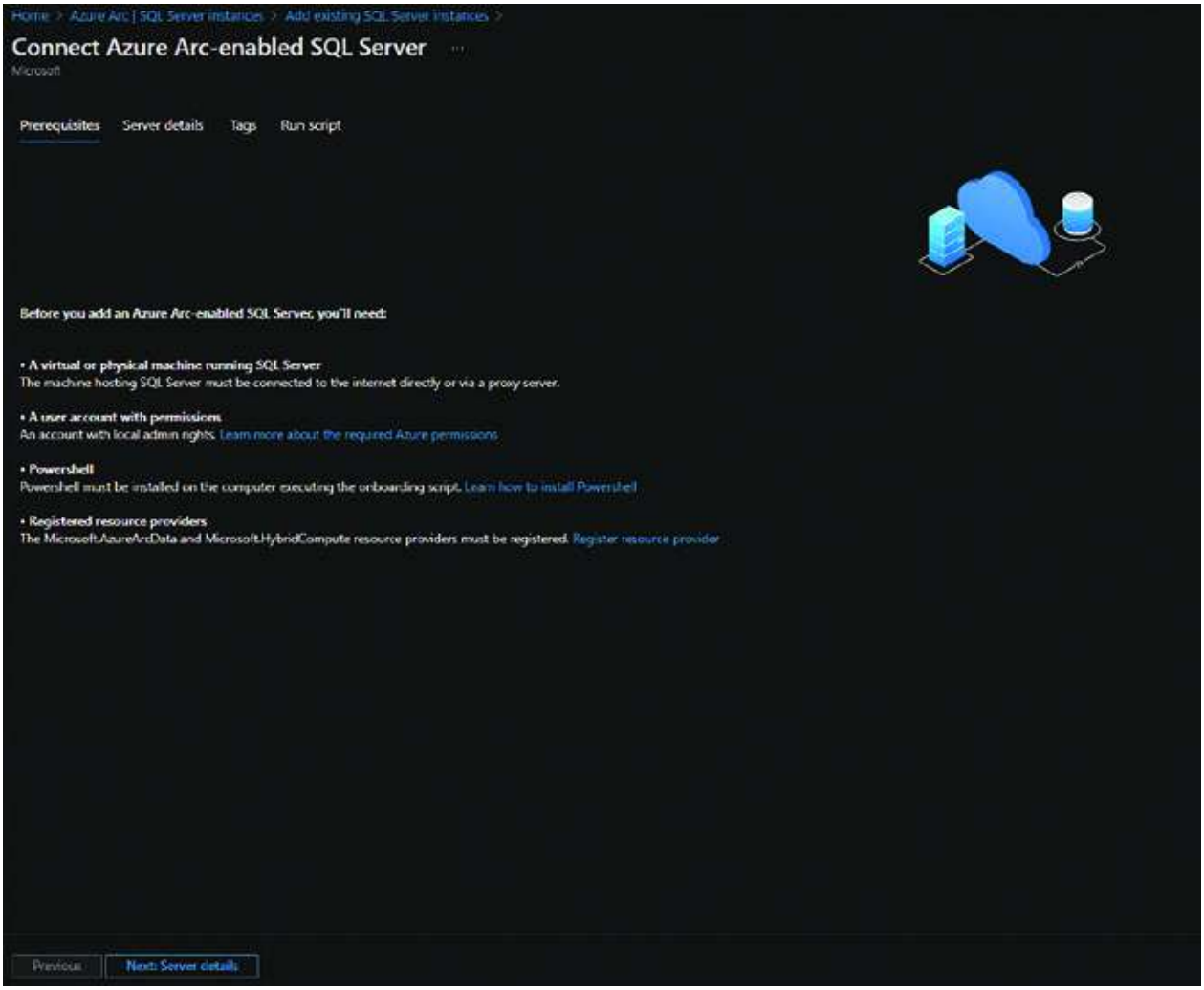


Figure 2.16: Important points before you connect SQL Server instance to Azure Arc.

Then provide the details like you did earlier, that is the resource group in Azure where you want this resource to be listed in Azure, and the Operating system details, so specific script can be provided as shown in Figure

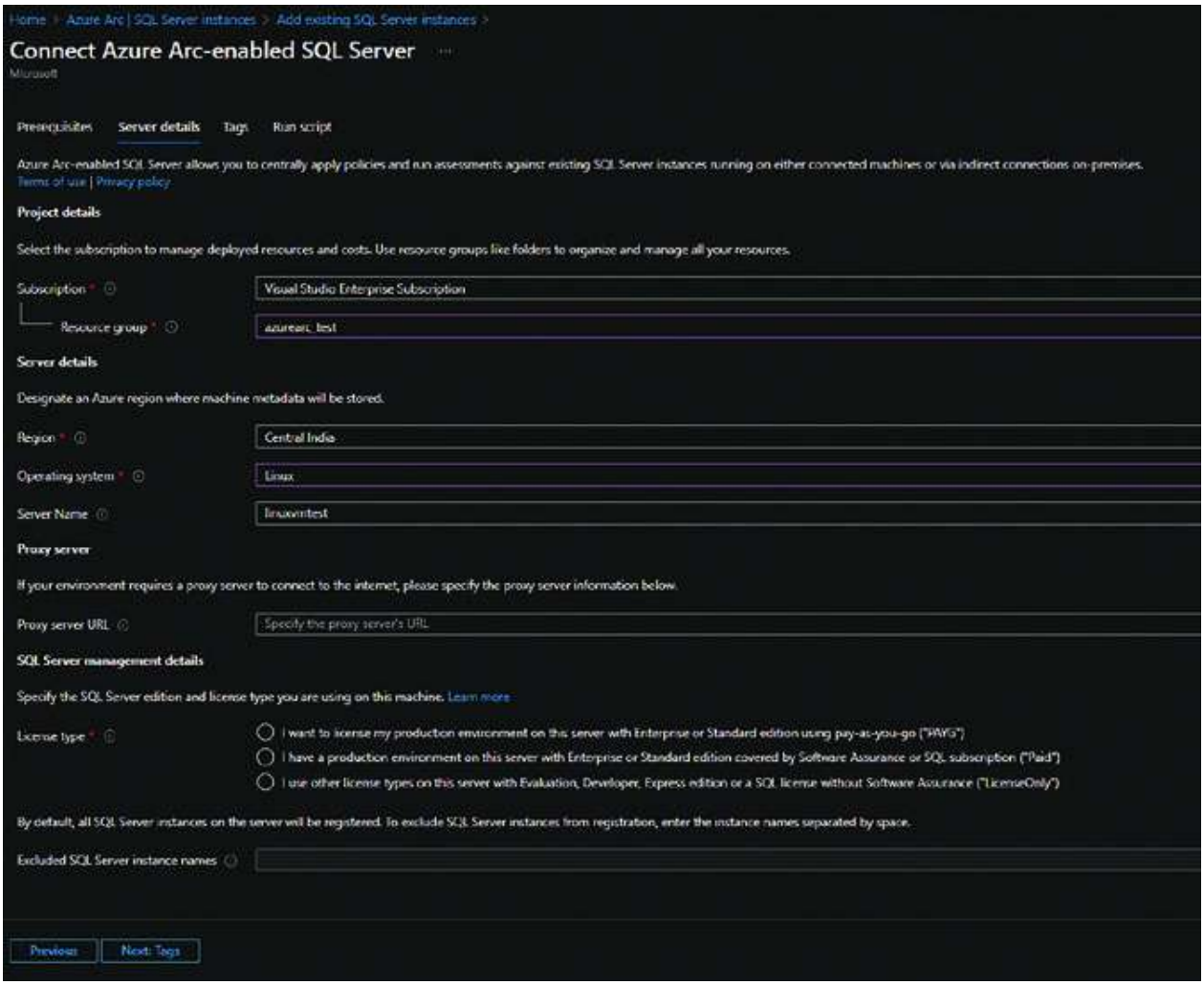


Figure 2.17: Providing SQL Server details to connect to Azure Arc

You now have the option to download the script. For the complete code, you can refer to the GitHub repository: Here is a summary of what the script does:

Installs the Az CLI client.

Registers the Microsoft.AzureArcData resource provider.

Checks if the current machine is Arc-joined; if not, it joins the machine by installing the Azure-connected machine agent.

Finally, installs the Azure extension for SQL Server

Once you run the script on the host and it completes, you can then see your SQL Server enlisted in Azure Arc, and your SQL Server is Azure Arc enabled SQL Server.

amvin@linuxvmtest:~$ sudo bash ./RegisterSqlServerArc.sh
Do you wish to install the Azure CLI? (y/N) yes
Installing Azure CLI
..
..
Installing Azure extension for SQL Server. This may take 5+ minutes.
Command group 'config' is experimental and under development. Reference and support levels: https://aka.ms/CLI_refstatus
WARNING: The command requires the extension connectedmachine. It will be installed first.
WARNING: The installed extension 'connectedmachine' is in preview.
Azure extension for SQL Server is successfully installed. If one or more SQL Server instances are up and running on the server, Arc-enabled SQL Server instance resource(s) will be visible within a minute on the portal.
Newly installed instances or instances started now will show up within an hour.

Figure 2.18 shows SQL Server instances which is Azure Arc-enabled.

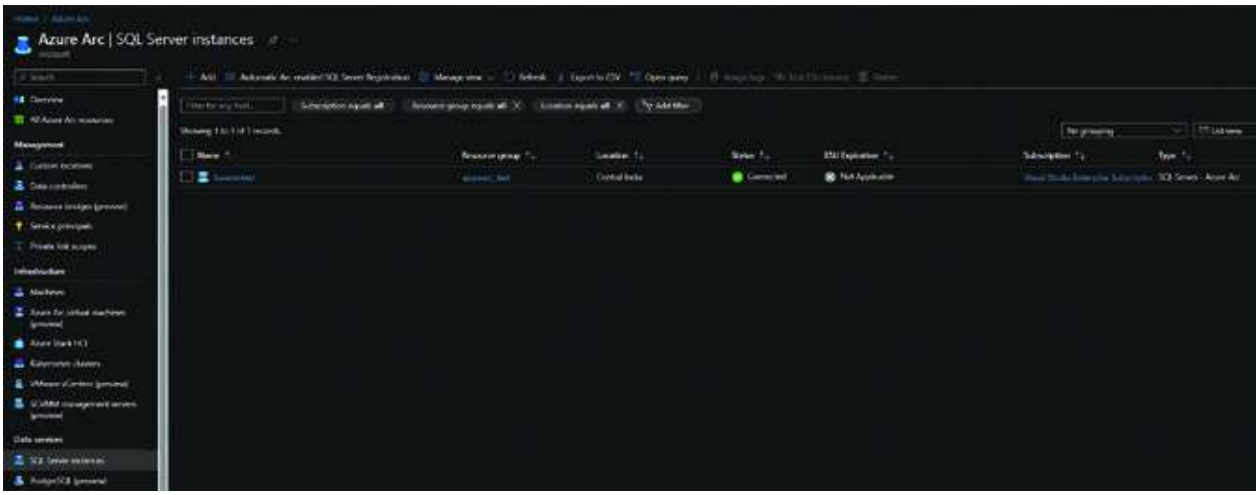


Figure 2.18: Portal showing SQL Server instance that is Azure Arc-enabled

Further, Figure 2.19 shows the details visible for the SQL Server instance that is Azure Arc enabled.

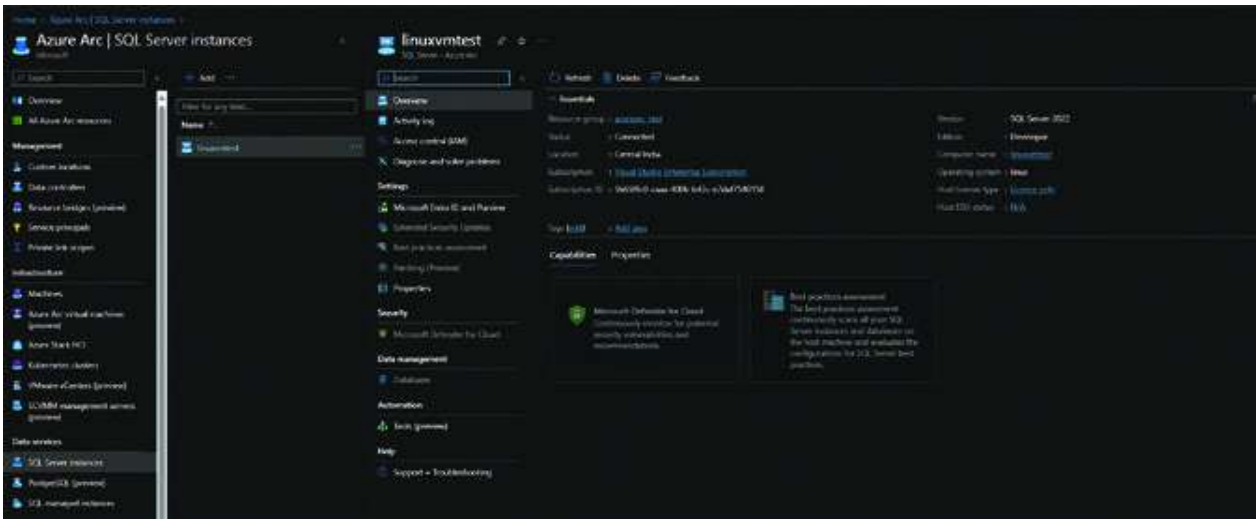


Figure 2.19: Azure Arc enabled SQL Server on Linux instance details.

Similarly, if you add an on-premises or non-Azure cloud-based SQL Server on a Windows machine to Azure Arc, then you will notice the options, such as best practice assessment, patching, and other enabled as they are supported for SQL Server instances on Windows as shown in Figure

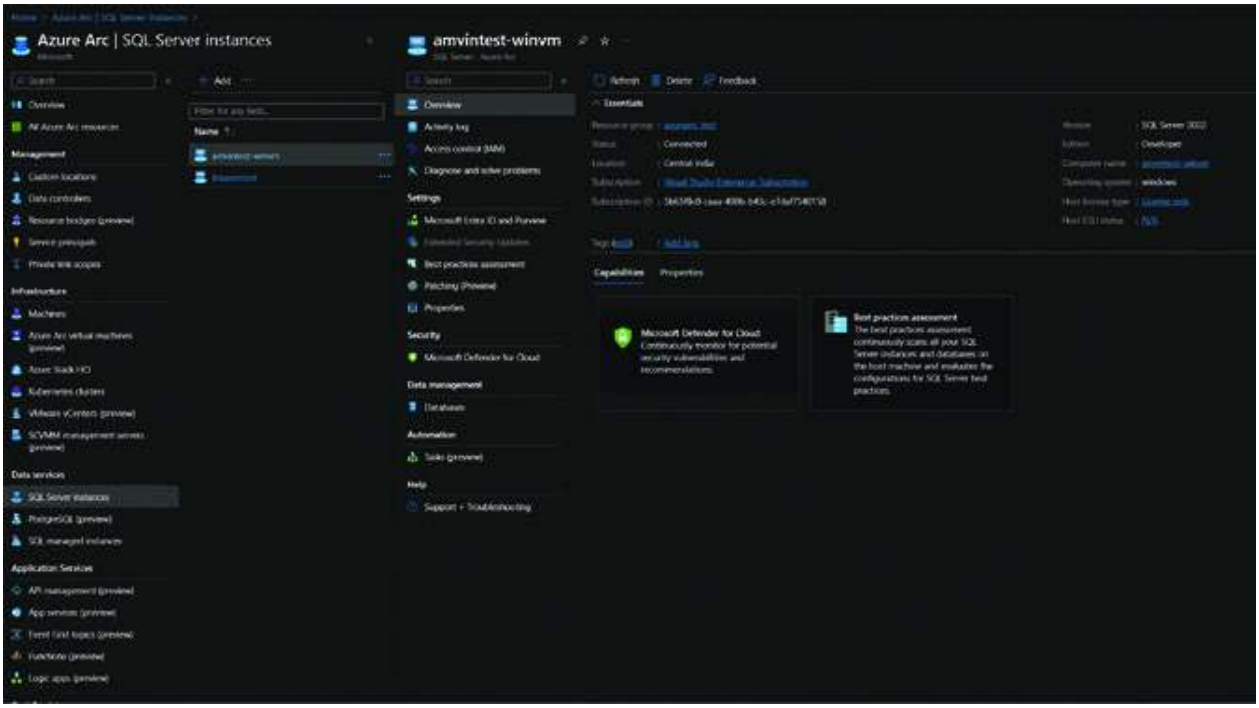


Figure 2.20: Azure Arc enabled SQL Server on Linux instance details.

If you choose to change the license type for SQL Server on Linux, you can do that by clicking on the license type option and selecting the Pay-as-you-go option as shown in Figure

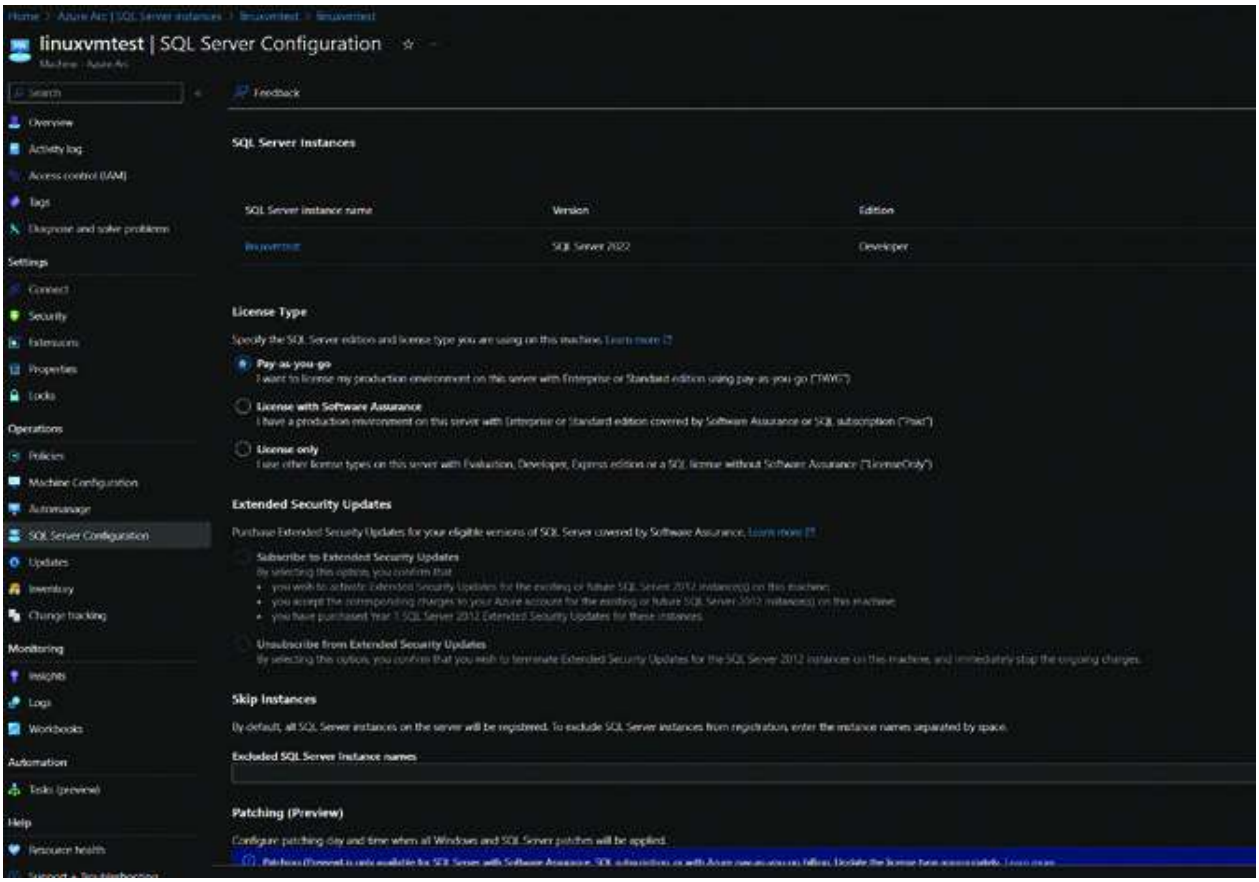


Figure 2.21: License Types available for Azure Arc enabled SQL Server

Once you hit the save button on the previous page where the license type is changed and set to this setting shows up in the portal, as shown here in Figure

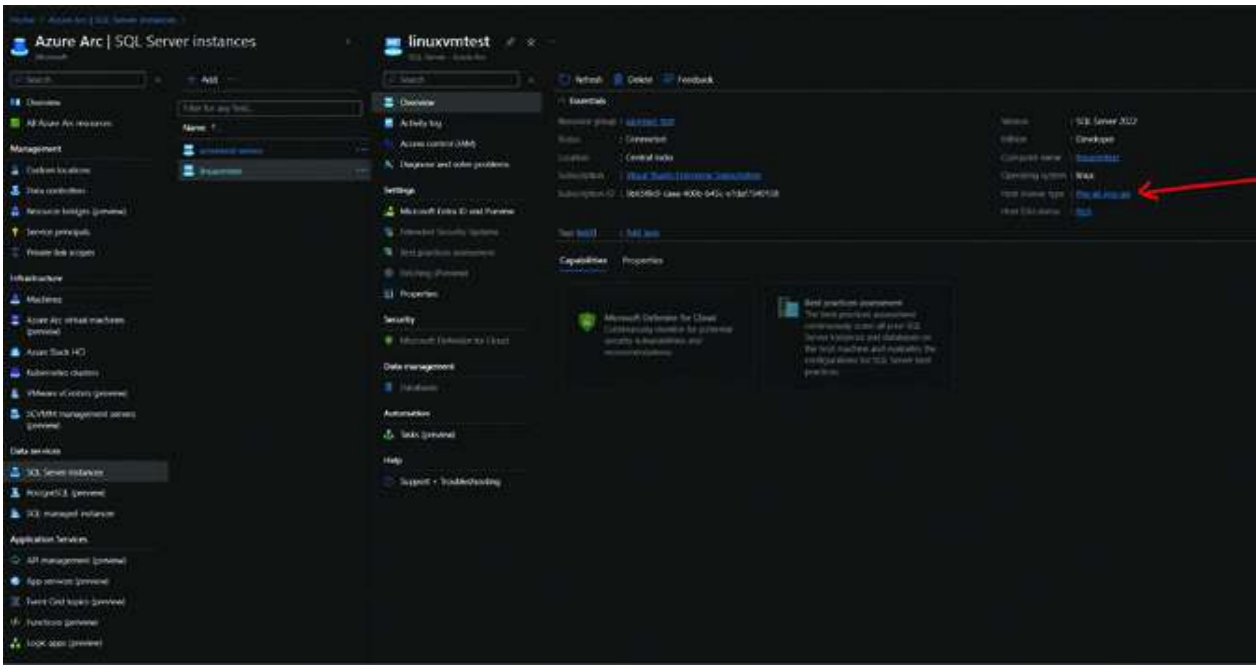


Figure 2.22: License type updated to PAYG

Azure arc-enabled Data Services

Azure Arc allows you to connect your on-premises and non-Azure cloud resources to Azure. This gives you access to some of the benefits of Azure, such as security, scalability, and manageability.

But what if you want to run Azure-specific data services on your non-Azure resources? For example, what if you want to run Azure SQL Managed Instance, which is a fully managed, up-to-date platform as a service that provides the broadest SQL Server engine compatibility?

Azure Arc-enabled data services make this possible. These services are managed by Microsoft, but deployed on your choice of environment outside of Azure. This gives you the flexibility to run Azure data services where you need them, without having to manage them yourself.

As of this writing, the following two Azure Data services are available:

Azure SQL Managed Instance

Azure Arc-enabled PostgreSQL (preview)

This book focuses on SQL Server; so we will look at Azure Arc-enabled SQL Managed Instance. We will first discuss how to bring Azure SQL Managed Instance outside of Azure to your choice of deployment. Then, we will see how to get started with Azure Arc-enabled SQL Managed Instance.

With Azure Arc, you onboarded and connected your on-premises and non-Azure cloud resources to Azure, which helps you to get some of the benefits of Azure for your non-Azure resources.

What if you wanted to run Azure-specific data services in a non-Azure environment of your choice? For instance, consider running Azure SQL Managed Instance, a fully managed, up-to-date platform as a service that offers the broadest SQL Server engine compatibility, within your own datacenter. Azure Arc-enabled data services can help you achieve and enable exactly this scenario.

As of writing this book, the following two Azure Data services are available:

SQL Managed Instance

Azure Arc-enabled PostgreSQL (preview)

Though this book focuses on SQL Server, here, we will look at Azure Arc-enabled SQL Managed Instance. Azure Arc-enabled SQL Managed Instance is a fully managed, evergreen SQL Server instance that you can deploy on your choice of infrastructure; whether it's on-premises, in a hybrid environment, or in a multi-cloud environment. It offers all the benefits of Azure SQL Managed Instance, including automatic backups, high availability, disaster recovery, scaling, and management capabilities.

Figure 2.23 is an architecture diagram from the official Microsoft documentation, Quick architecture guide for Azure Arc-enabled data services (microsoft.com) that can help explain how this works outside Azure.

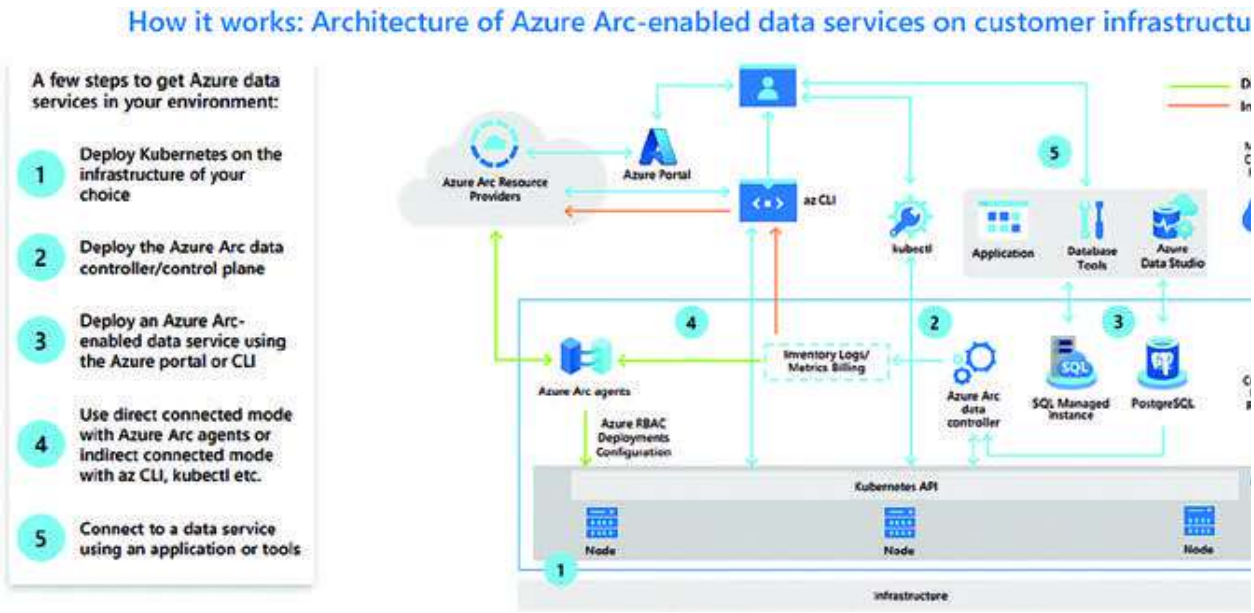


Figure 2.23: Azure Arc-enabled data services Architecture

As shown in the diagram, Azure Arc-enabled data services are deployed outside the Azure environment using a Kubernetes cluster and an Azure Arc data controller. The Azure Arc data controller provides the control plane you need to deploy and manage Azure data services anywhere.

You can deploy a Kubernetes cluster on your choice of hardware, such as Red Hat OpenShift, VMware Tanzu Kubernetes Grid, Canonical Charmed Kubernetes, SUSE Rancher, or public cloud provider-specific Kubernetes platforms like Google Kubernetes Engine (GKE) and Amazon Elastic Kubernetes Service (EKS). Once you have deployed a Kubernetes cluster, you can attach it to Azure using Azure Arc-enabled Kubernetes. This will allow you to deploy an Azure Arc data controller, which provides the control plane for managing Azure data services on your non-Azure platform. Let's take a look at a hands-on demo to see how all of this works. This will help to explain it in more detail.

To follow this demo, you will need the following prerequisites:

A Windows or Linux-based client machine with the following tools installed:

az cli

kubectl

A Kubernetes cluster that is connected to Azure Arc. For this demo, I will create an Azure Kubernetes Service (AKS) cluster and then connect it to Azure Arc. You can use any other Kubernetes cluster that is compatible with Azure Arc.

On the Windows client machine, we will run the following command to install the Azure CLI:

winget install -e --id Microsoft.AzureCLI

This command uses the Windows Package Manager (winget) to install the Azure CLI. Once the installation is successful, we can run the following command to verify the installation:

az --version

This command will show the version of the Azure CLI that is installed; Figure 2.24 here is a sample for the same.

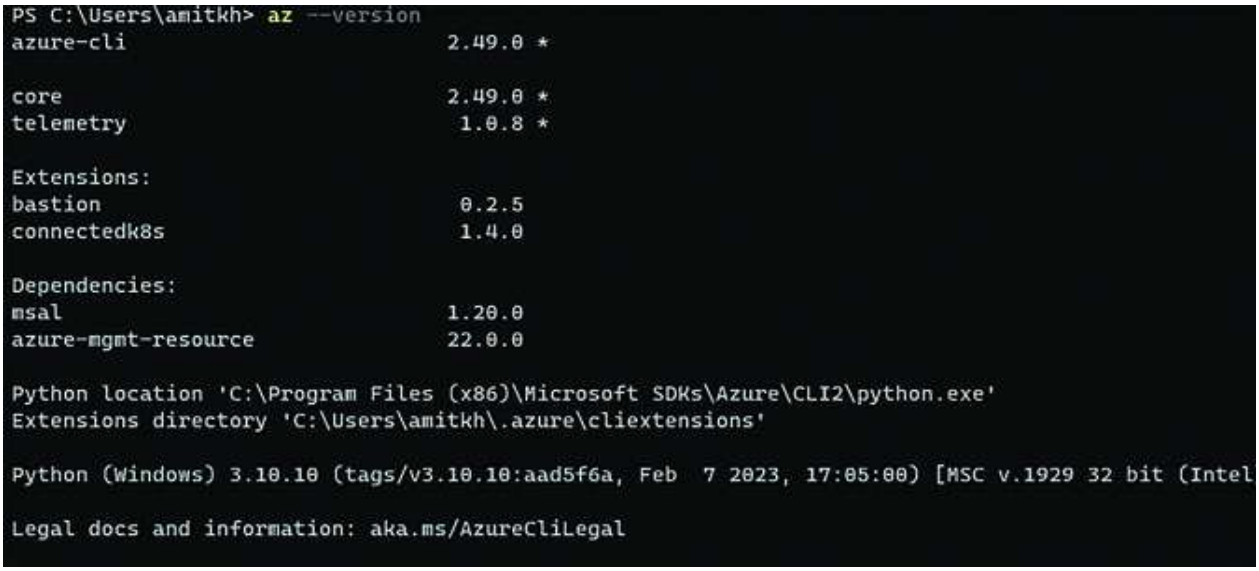


Figure 2.24: az –version sample output

Now, let's install kubectl, which is the client required to connect and work with Kubernetes. We can install kubectl on the same Windows client using the following command:: winget install -e --id Kubernetes.kubectl

It is time to create the Kubernetes cluster, and you can create the Kubernetes cluster using the az cli. Here are the steps to create the managed Azure Kubernetes cluster

Login to Azure, if not already done:

az login

Create the Azure Kubernetes cluster using the command:

az aks create --resource-group azurearc_test --name testk8samvin --node-count 2 --generate-ssh-keys –node-vm-size Standard_D4ads_v5

You need to provide the resource group, name of the cluster, node count and node-vm-size based on your requirement.

Once the Kubernetes cluster is created, connect to the cluster using the command:

az aks get-credentials --resource-group azurearc_test --name testk8samvin

Once connected, you can verify your connection by running general kubectl commands like the kubectl get nodes and kubectl get Figure 2.25 shows the sample for the same.

```
C:\Users\amitkh>kubectl get all
NAME                 TYPE        CLUSTER-IP   EXTERNAL-IP   PORT(S)   AGE
service/kubernetes   ClusterIP   10.0.0.1     <none>        443/TCP   124m

C:\Users\amitkh>kubectl get nodes
NAME                                STATUS   ROLES   AGE    VERSION
aks-agentpool-23585954-vmss000000   Ready    agent   123m   v1.26.6
aks-agentpool-23585954-vmss000001   Ready    agent   123m   v1.26.6
```

Figure 2.25: Kubectl commands showing the resources and nodes of the kubernetes cluster

Ensure that the following extensions are registered on the Azure subscription; if not, then please use the Azure portal and register them as shown for the subscription that you intend to use.

Microsoft.Kubernetes
Microsoft.kubernetesConfiguration

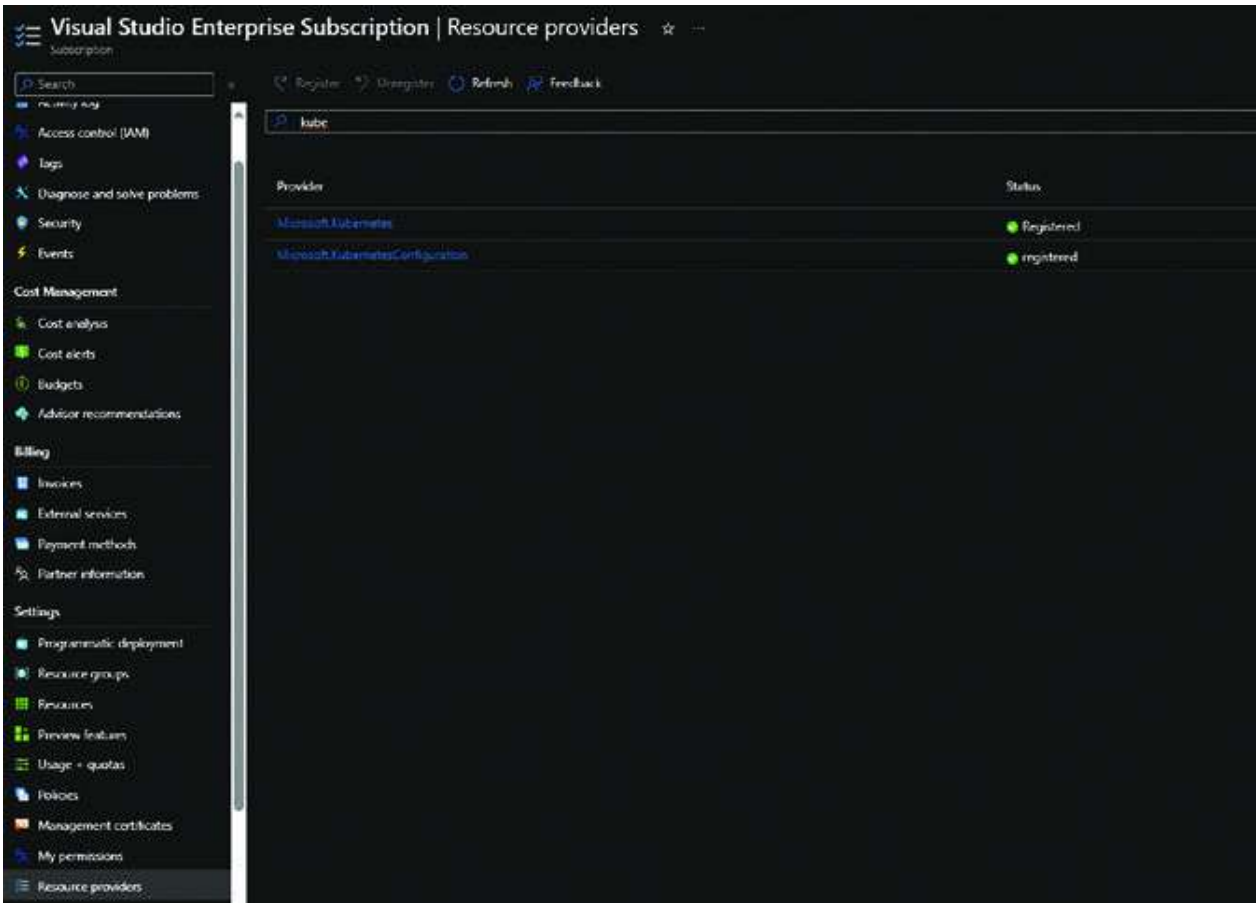


Figure 2.26: List of the resource providers to be registered for the Azure subscription

It’s time to connect the previously created Azure Kubernetes service-based cluster to Azure Arc; to accomplish this task run the below command:

az connectedk8s connect --resource-group azurearc_test --name testk8samvin

Once this is successful, you will notice the shadow object with the same name Kubernetes cluster name created, this is the representation of the Azure Arc-enabled cluster as shown in Figure

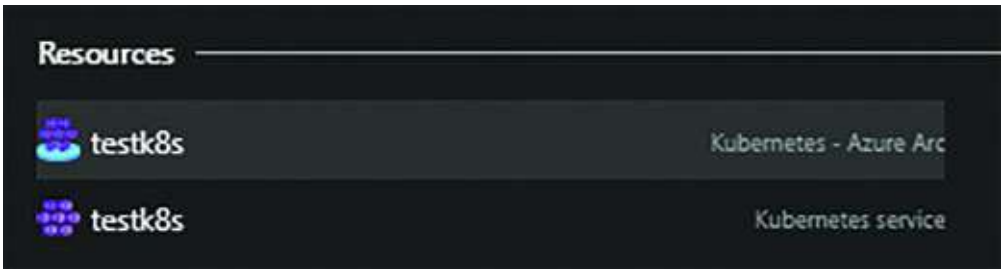


Figure 2.27: The shadow object of the Kubernetes cluster as seen in the portal

As you can see there is an object created in the portal that represents the cluster testk8s for Azure Kubernetes Service, and then another object with the same name called testk8s but the resource type is Kubernetes-Azure Arc. You can also see the testk8s enlisted as a Kubernetes cluster on the Azure Arc blade as shown in Figure

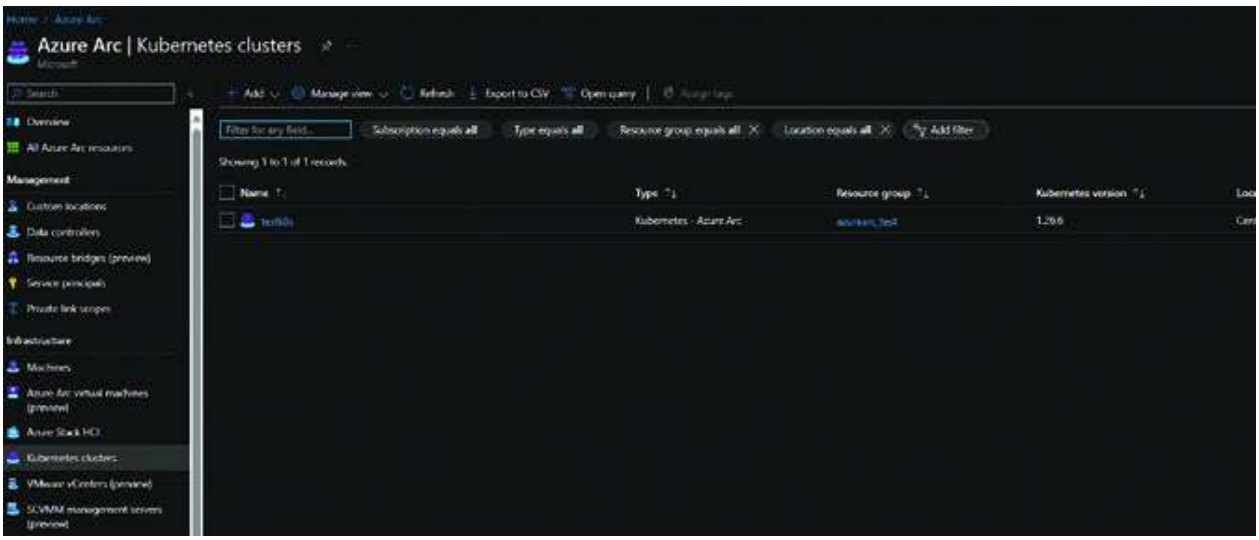


Figure 2.28: Azure Arc blade in the portal showing the testk8s cluster

After the Kubernetes cluster is successfully connected to Azure Arc, when you run a kubectl command to enlist the namespaces, you will see two new namespaces created in your Kubernetes cluster, called “azure-arc” and “azure-arc-release” that has resources running on your cluster that helps

connect your Kubernetes cluster to Azure. If these namespaces are not created, that means there was an issue with the Kubernetes cluster connecting to Azure, so you might have to go back and redo steps if these namespaces are missing.

```
C:\Users\amitkh>kubectl get namespaces
NAME                STATUS   AGE
azure-arc           Active   121m
azure-arc-release   Active   126m
calico-system       Active   144m
default             Active   145m
kube-node-lease     Active   145m
kube-public         Active   145m
kube-system         Active   145m
tigera-operator     Active   144m
```

We have now completed the task of connecting the Kubernetes cluster to Azure Arc; the next step is to create the Azure Arc data controller, which is the control plane that enables you to deploy the Azure data services. You need to create the Azure Arc data controller in the same resource group as the Kubernetes cluster, so in the Azure portal, under the marketplace place, search for the Azure Arc data controller as shown in Figure

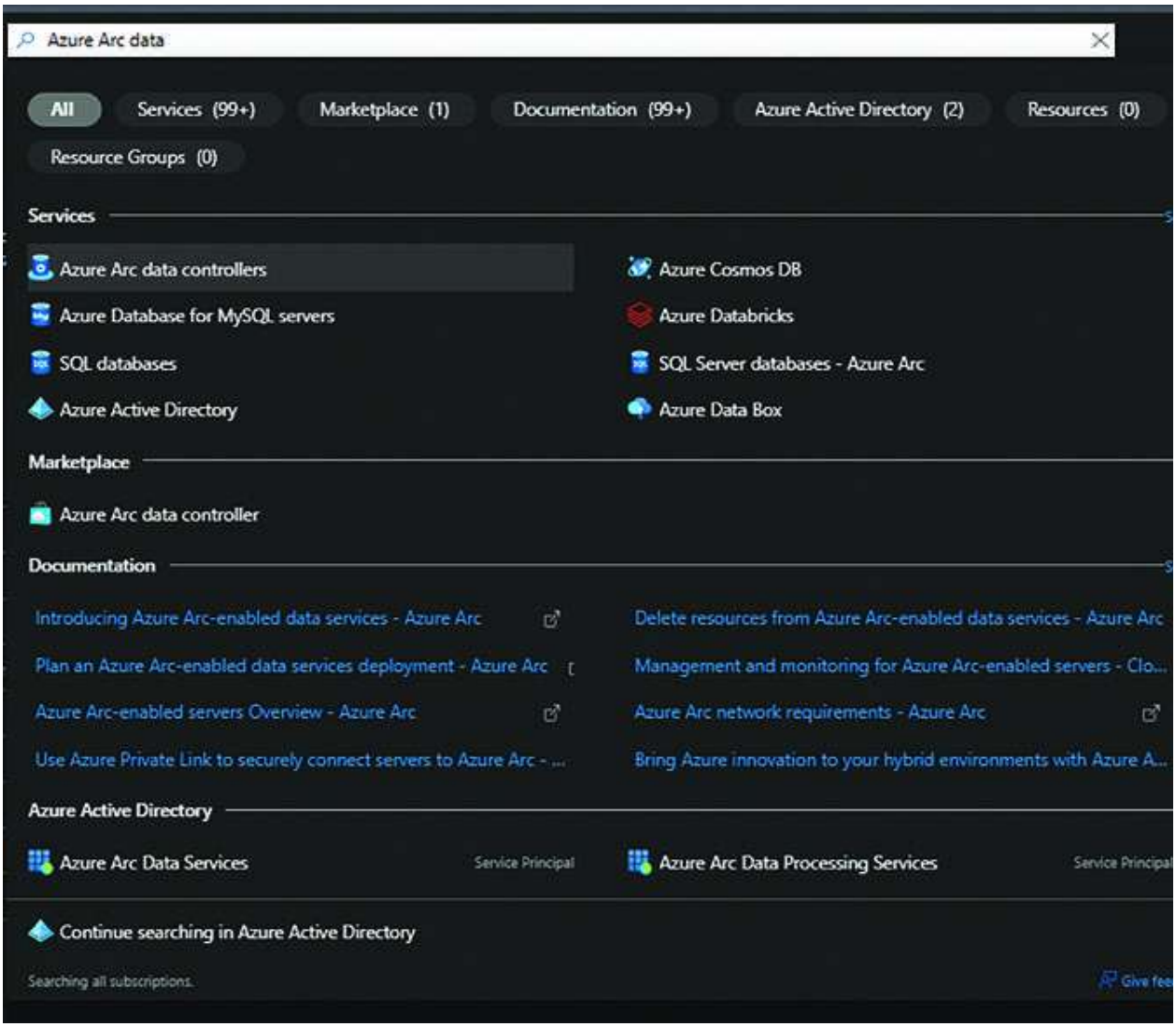


Figure 2.29: Locating the Azure Arc data controller in Azure portal

Select Create Azure Arc data controller as shown in Figure Then choose the direct connectivity mode option.

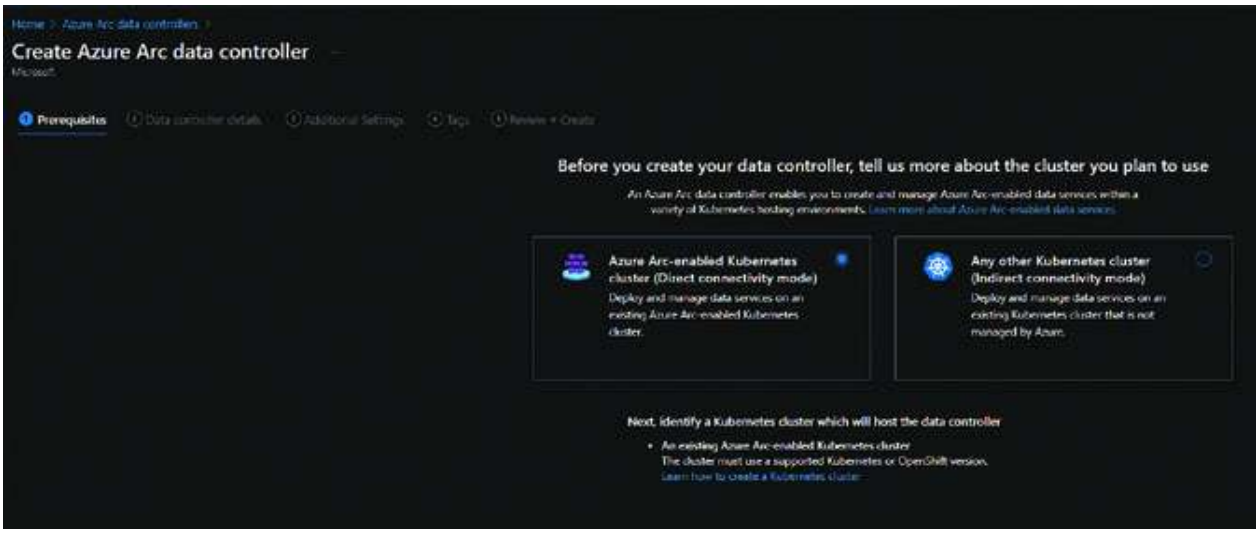


Figure 2.30: Choosing the Direct connectivity mode for the Azure Arc data controller

In the next screen, you will have to provide the data controller name and a custom location. Here, the custom location represents the new namespace that you want to create on your existing Kubernetes cluster that will be used to create the Azure data service resources.

When you click on the new custom location, you get the option to also choose the cluster. You can have only one data controller on one Kubernetes cluster, in this case, the custom location (namespace name) is given as dcsqlmi and the cluster selected from the drop-down is testk8s as shown in Figure

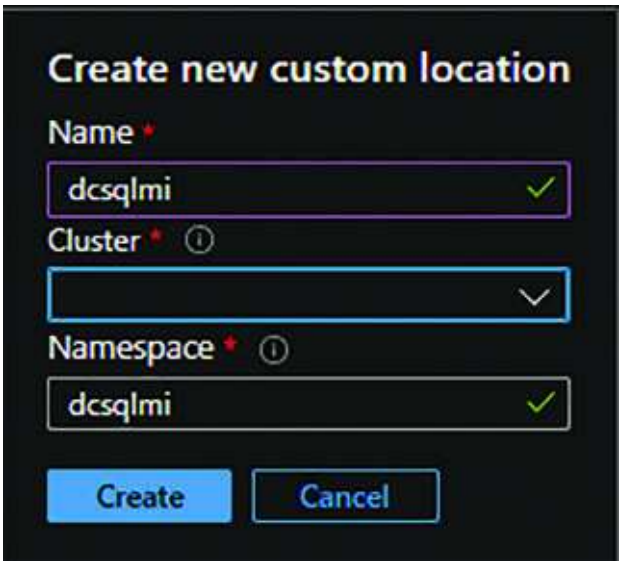


Figure 2.31: Choosing the cluster name and namespace.

In the “Kubernetes configuration template” you need to select the template from the drop-down that matches your Kubernetes distributions. There are multiple options as shown in Figure if you chose to use the Google Kubernetes engine initially to connect to Azure, then you should choose the azure-arc-gke option. If you chose AWS Elastic Kubernetes Service, then you should choose azure-arc-eks. For this demo, Azure Kubernetes Service with default storage is used. Here, “azure-arc-aks-default-storage” is selected from the options.

Figure 2.32: Multiple Kubernetes configuration template choices

Then we will provide a username and password to upload the metrics and log dashboard credentials which will also need the log analytics workspace. In this case, when creating the data controller, log analytics is not enabled. You can see all the details needed in Figure

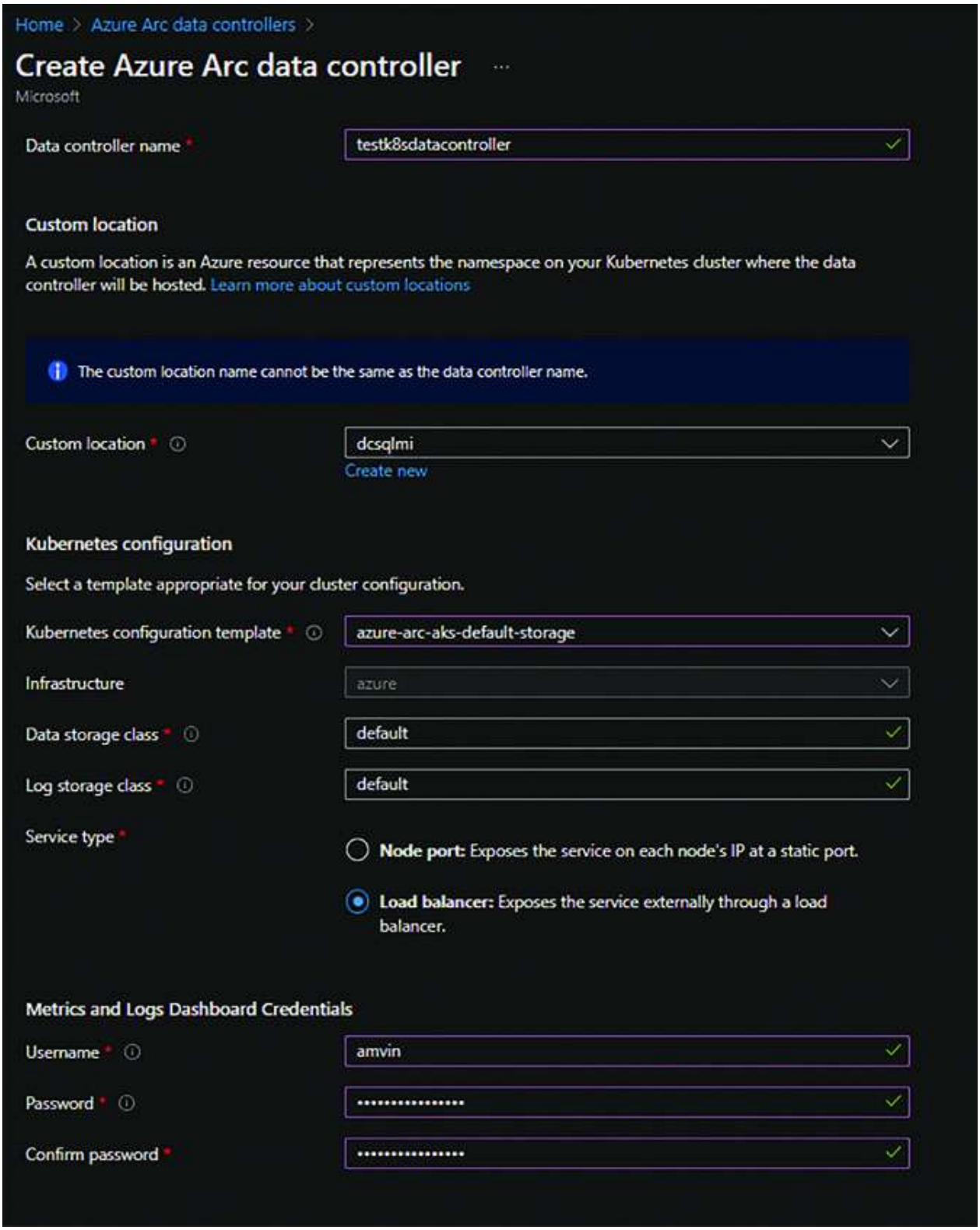


Figure 2.33: Azure Arc data controller creation page

Once the deployment of the data controller is successful in the portal, you can verify that the name of the new namespace is “dcsqlmi”. You should see this created on the Kubernetes cluster and you can enlist the namespace using the same kubectl get namespace command that is:

```
C:\Users\amitkh>kubectl get namespaces
NAME              STATUS   AGE
azure-arc         Active   5m55s
azure-arc-release   Active   8m48s
calico-system      Active   15m
dcsqlmi           Active   34s
default           Active   15m
kube-node-lease     Active   15m
kube-public       Active   15m
kube-system       Active   15m
tigera-operator     Active   15m
```

Once you confirm the namespace is created, you will have to check the state of the data controller and before proceeding, you need to ensure that the state shows as ready to verify the state of the data controller. Run the below kubectl command and initially, the state of the data controller is going to be as shown in Figure

```
C:\Users\amitkh>kubectl get datacontrollers -A
NAMESPACE   NAME                    STATE
dcsqlmi     testk8sdatacontroller   DeployingController

C:\Users\amitkh>kubectl get datacontroller -n dcsqlmi
NAME                    STATE
testk8sdatacontroller   DeployingController
```

Figure 2.34: The data controller state still showing as deploying

You can query the Kubernetes cluster to understand what is happening in the background when the data controller is being deployed. To check the logs, you can run the following command. You can also check the output given.

```
kubectl describe datacontroller -n dcsqlmi
..
..
Status:
Azure:
Upload Status:
Logs:
Last Upload Time:  0001-01-01T00:00:00Z
Message:
Metrics:
Last Upload Time:  0001-01-01T00:00:00Z
Message:
Usage:
Last Upload Time:  0001-01-01T00:00:00Z
Message:
Last Update Time:        2023-09-12T14:56:05.134372Z
Observed Generation:     1
Running Version:         v1.22.0_2023-08-08
State:                   Ready
Events:
Type    Reason              Age    From           Message
----    ------              ----   ----           -------
Normal  Initializing        5m44s  datacontroller  Initiating Data Controller

Normal  DeployingController  5m44s  datacontroller  Initiating control plane deployment
```

Normal Creating 4m datacontroller Preparing for DataController dcsqlmi/testk8sdatacontroller.
Normal Ready 4s datacontroller DataController dcsqlmi/testk8sdatacontroller is ready

It would take about 5-10 minutes for the data controller to be in the ready state. Proceed to the next step only after you see the data controller is in the ready state as shown in Figure

```
C:\Users\amitkh>kubectl get datacontroller -n dcsqlmi
NAME                    STATE
testk8sdatacontroller   Ready
```

Figure 2.35: Data controller state now in the ready state

Once the data controller is deployed and shown in a ready state with the namespaces created in the Kubernetes cluster, you are now ready to install your choice of Azure Data service. For this demo, we are going to deploy Azure Arc -enabled SQL Managed Instance. To do that, search for the managed instance on the Azure portal and then choose the “SQL managed instances -Azure Arc” option as shown in Figure

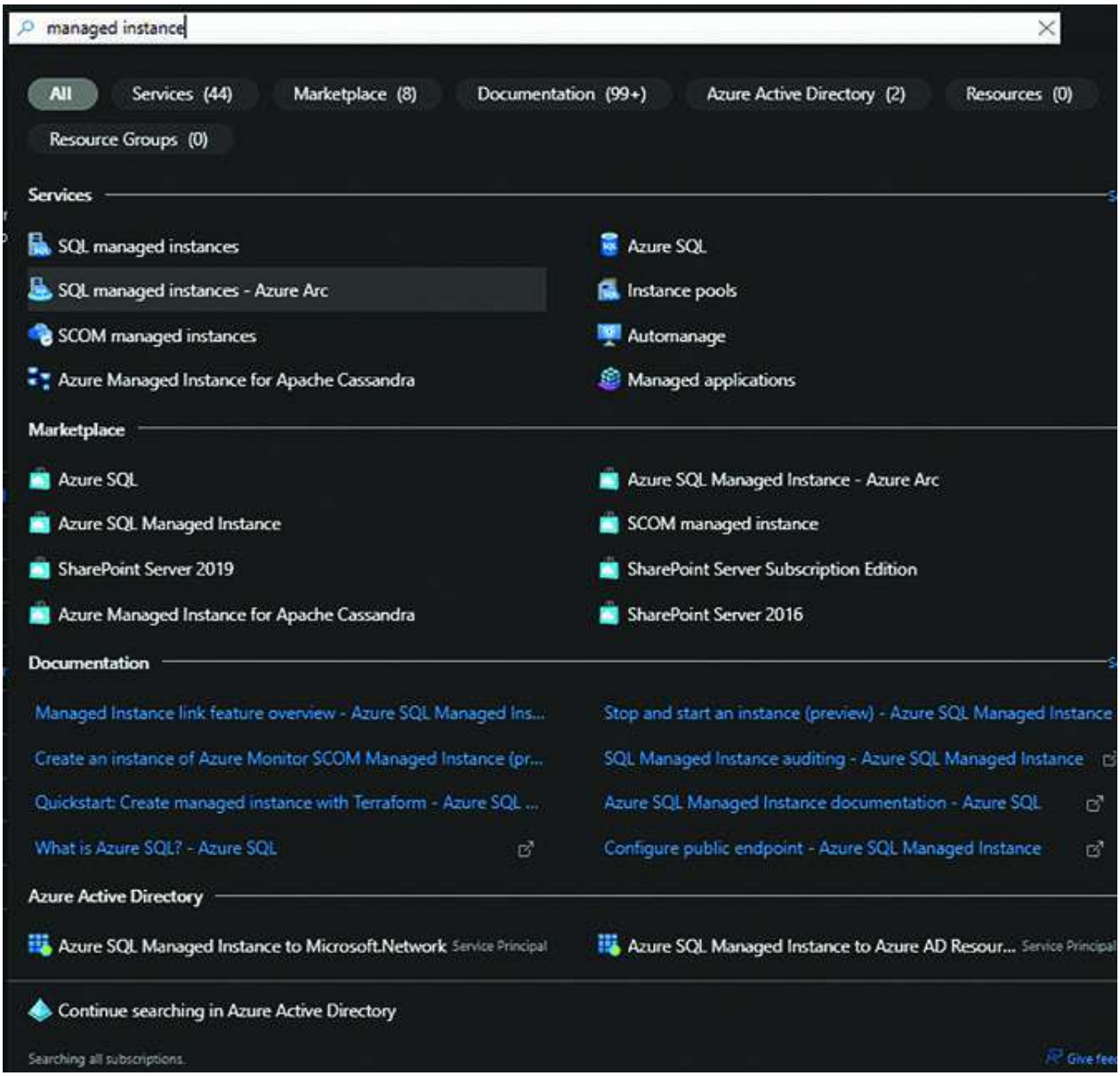


Figure 2.36: Creating the SQL Managed Instance via the portal.

Create a new SQL MI instance and provide the details as follows:

Please ensure you choose the same resource group that was used in previous steps when creating and joining the Kubernetes cluster to Azure. In this case, use the “azurearc_test”

The custom location is the namespace you used to create the data controller in the Kubernetes cluster. In this demo, the value of the custom location is

“dcsqlmi”

You can configure the compute and storage based on your requirements. When you click the “configure compute+storage” option, you can also choose the licensing of SQL Server to be used as follows. The compute and storage values depend on the size of your Kubernetes cluster that you choose when deploying the cluster.

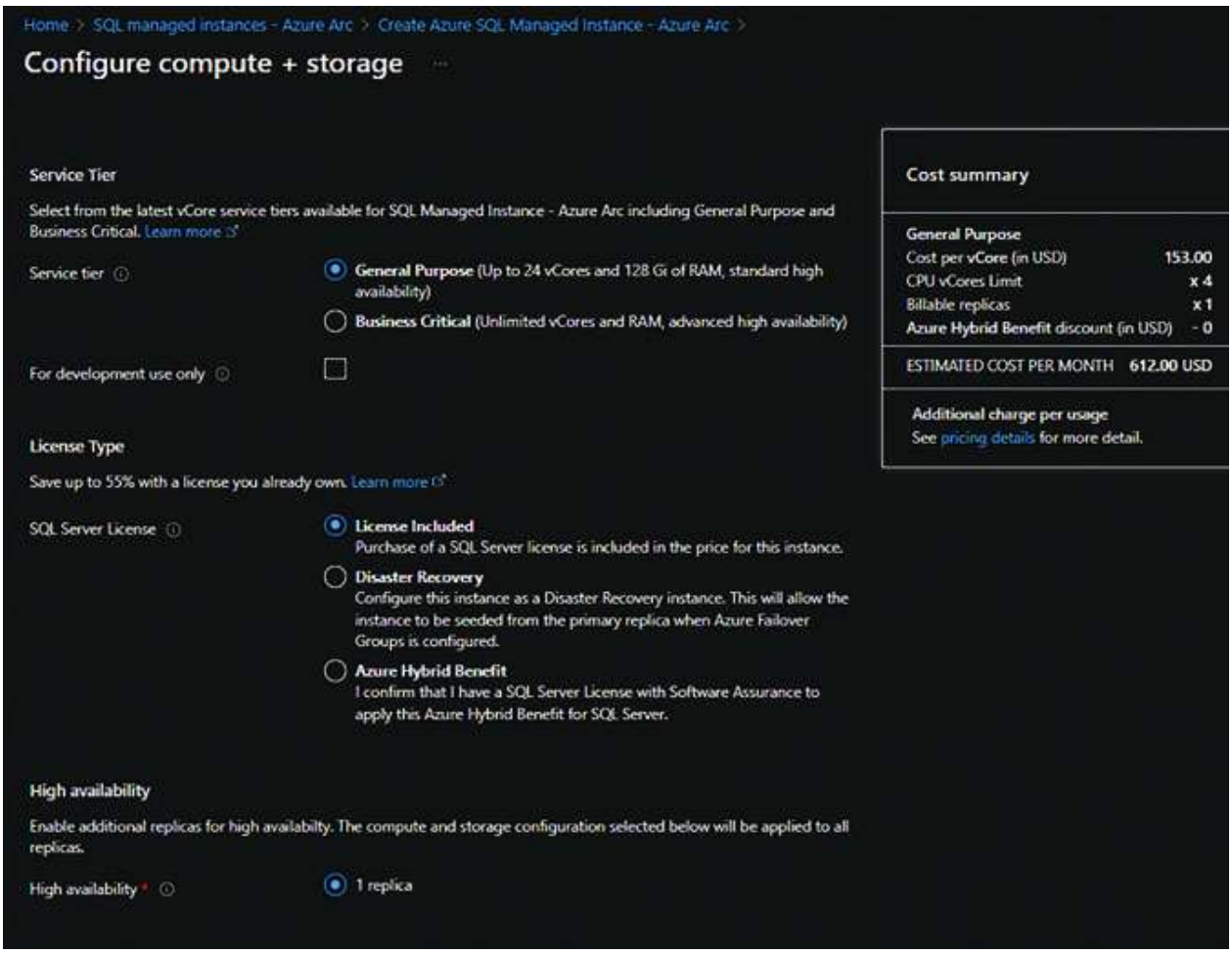


Figure 2.37: Configuring compute storage and other details for the Azure SQL Managed Instance.

Choose the High availability options and the resource limits that you intend to allocate to the Azure SQL Managed Instance as shown in Figure

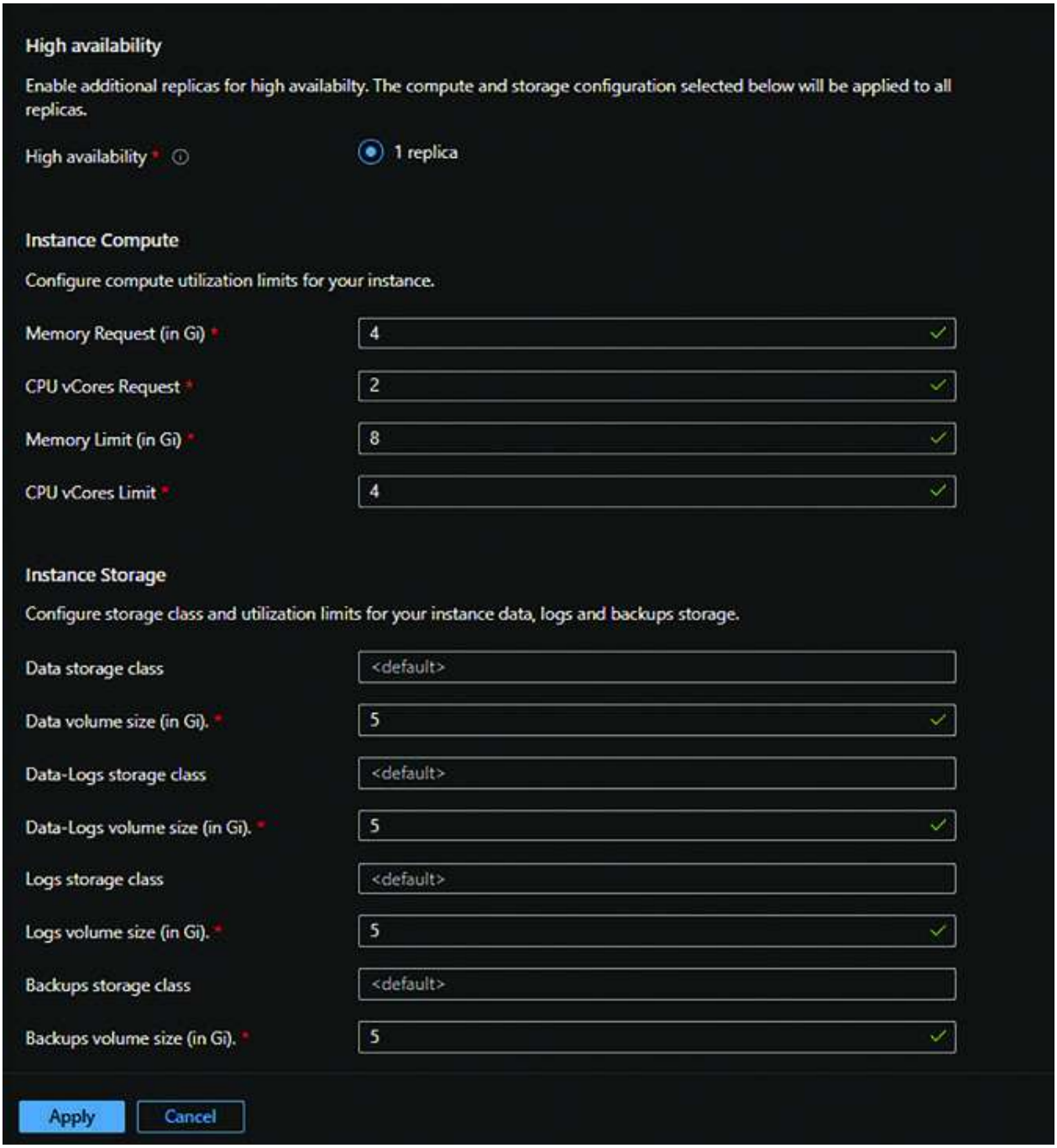


Figure 2.38: High Availability and resource limits specified for Azure SQL Managed Instance.

Lastly, you need to provide the service type as a load balancer to be able to connect to the SQL MI externally outside the Kubernetes cluster and provide

the administrator account details for the SQL user to be created using which you can log in to SQL MI for the first time. For details, refer Figure

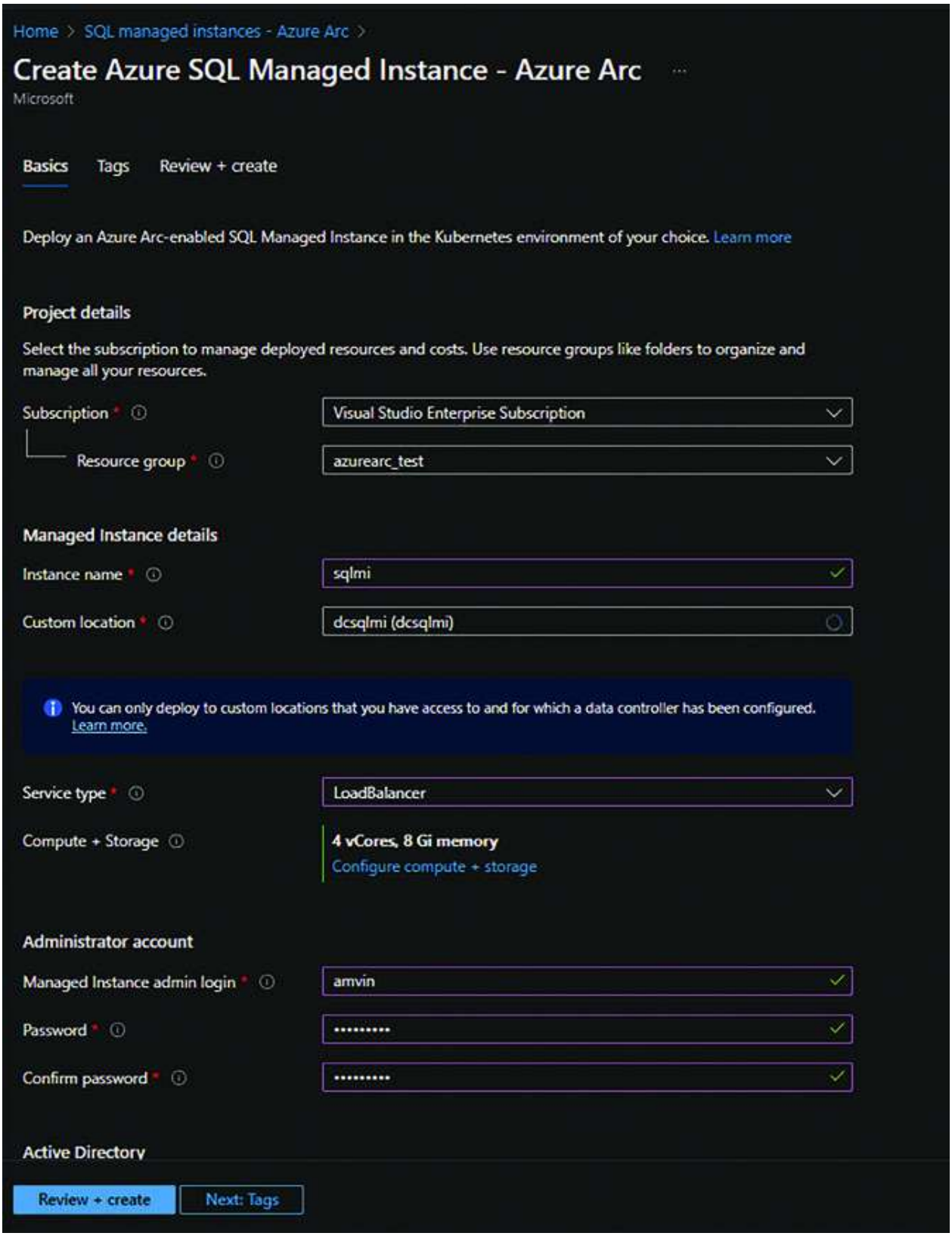


Figure 2.39: Setup load balancer to connect to Azure SQL MI

When you are ready, hit the Review+create button and the Azure Arc-enabled SQL Managed Instance deployment starts. Once it completes, you should see a message in the portal as shown here in Figure

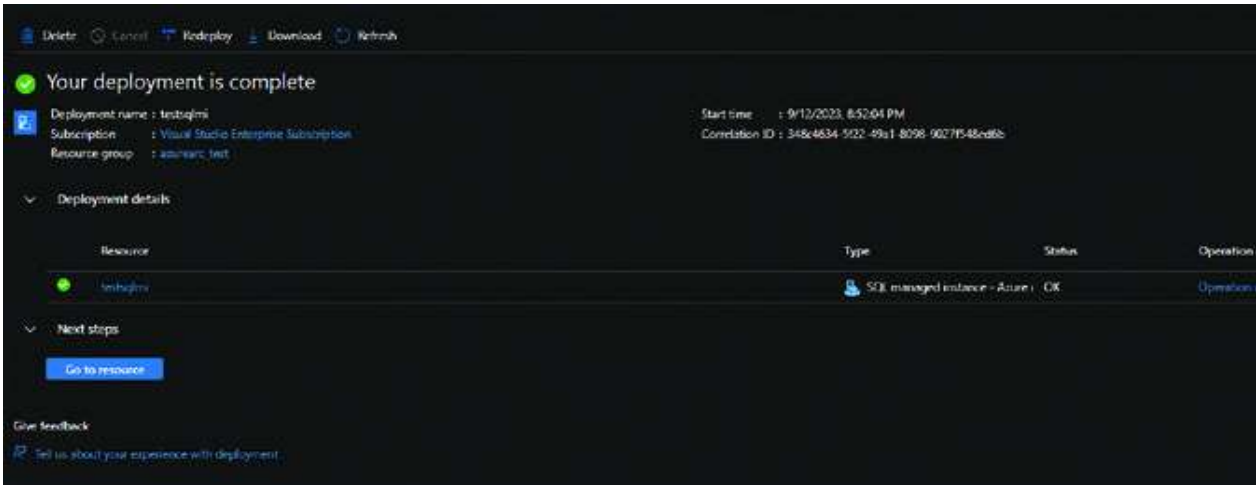


Figure 2.40: Successful deployment of the Azure SQL MI as seen in the portal.

Before you can connect to the SQL-managed instance running on your Kubernetes cluster, you need to ensure the deployment is ready. You can run the following kubectl command to check the status of the sqlmi resource under the dcsqlmi namespace as given here.

kubectl describe sqlmi -n dcsqlmi
Name: testsqlmi
Namespace: dcsqlmi
Labels:
management.azure.com/resourceProvider=Microsoft.AzureArcData
..

..
Events:
Type Reason Age From Message
---- ------ ---- ---- -------
Normal Creating 5m25s datacontroller Preparing for SqlManagedInstance dcsqlmi/testsqlmi.
Normal Creating 5m20s datacontroller Creating StatefulSet testsqlmi in SqlManagedInstance testsqlmi

Normal Ready 59s datacontroller StatefulSet testsqlmi in SqlManagedInstance testsqlmi is ready
Normal Creating 59s datacontroller Creating StatefulSet testsqlmi-ha in SqlManagedInstance testsqlmi
Normal Ready 9s datacontroller StatefulSet testsqlmi-ha in SqlManagedInstance testsqlmi is ready

To get the connection details and status of the SQL MI, run the command given, as you can see from the output below the status of SQL MI instance called testsqlmi is ready, which means we can connect to the instance now using the IP address 4.157.65.71.

C:\Users\amitkh>kubectl get sqlmi -n dcsqlmi
NAME STATUS REPLICAS PRIMARY-ENDPOINT DESIRED-VERSION RUNNING-VERSION AGE
testsqlmi Ready 1 4.157.65.71,1433 v1.22.0_2023-08-08 6m1s

You can use your choice of SQL Server tools such as the SQL Server Management Studio (SSMS) or Azure Data Studio (ADS) to connect to the Azure Arc -enabled SQL Managed instance and work with it as you would normally work with any SQL Server instance deployed on-premises. As you

can see in Figure we are using the SSMS tool and the account amvin that was created during the Azure SQL MI creation in previous steps:

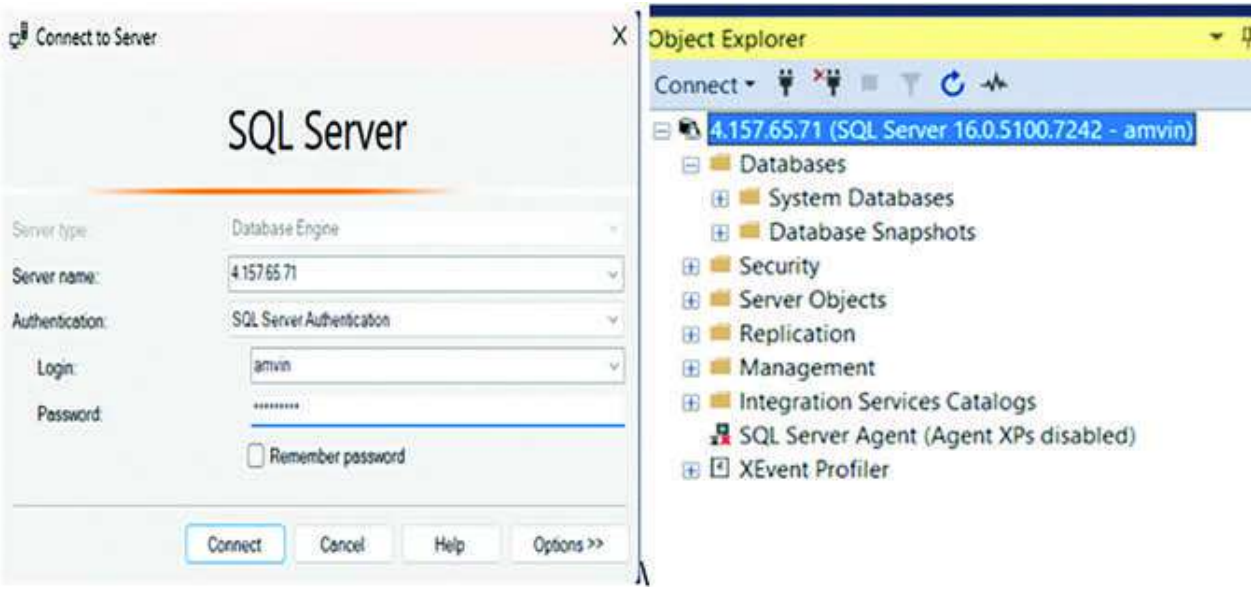


Figure 2.41: Connecting to the Azure SQL MI using SQL Server Management Studio.

This concludes the section on Azure Arc. We have seen how you can use different flavors of Azure Arc, such as the Azure Arc -enabled servers, Azure Arc -enabled Kubernetes cluster, Azure Arc-enabled SQL Server, and Azure Arc -enabled data services, to create a true hybrid environment. In a hybrid environment, you choose and manage the location and hardware of your deployments, but you can manage them all through a single pane of glass in Azure. You can also run Azure data services of your choice outside Azure on your choice of deployments.

Conclusion

We looked at the overall SQL Ecosystem and the different options available for deployment. We understood the key concepts like structured and unstructured data and workloads such as OLTP vs OLAP vs HTAP. We then looked at SQL Server On-Premises deployment option. Then we looked at the Azure Arc options that help you enable a true hybrid deployment that spans across on-premises and cloud infrastructure.

In the next chapter, we will look at the rest of the deployment options of SQL Ecosystem which are SQL Server on Linux, Containers, and Azure SQL.

Additional Resources

Columnstore indexes: Overview - SQL Server | Microsoft Learn:

SQL Server 2022: Hardware & software requirements - SQL Server | Microsoft Learn

Considerations for installing SQL Server using SysPrep - SQL Server | Microsoft Learn

SQL Server 2022—Pricing | Microsoft

Overview of the Azure Connected Machine agent - Azure Arc | Microsoft Learn

Azure Arc overview - Azure Arc | Microsoft Learn

Plan and deploy Azure Arc-enabled servers - Azure Arc | Microsoft Learn

Azure Arc-enabled SQL Server - SQL Server | Microsoft Learn

Quick architecture guide for Azure Arc-enabled data services (microsoft.com)

CHAPTER 3

Realms of SQL - Part 2

Introduction

In this chapter, we will continue our discussion on the different Realms of SQL and turn our attention to SQL Server on Linux. We will look at the architecture and try answering a few perennial questions:

Is SQL Server on Linux the same database engine as SQL Server that runs on Windows?

Can we use the same SQL Server tools to work with SQL Server on Linux as we do with SQL Server on Windows?

Structure

In this chapter, we will cover the following topics:

SQL Server on Linux Architecture

Getting Started with SQL Server on Linux

Getting Started with SQL Server Containers and Deployment on Kubernetes Clusters

Understanding the Azure SQL Family Starting with SQL Server Azure Virtual Machines

Azure SQL Database

Azure SQL Managed Instance

Basics of Azure SQL Edge

SQL Server on Linux and Containers

In this section, when we talk about SQL Server on Linux, we are referring to both SQL Server deployments on Linux servers (including virtual machines) and SQL Server containers. Also, it is necessary to mention that SQL Server on Linux or container-based deployments are fully supported for production workloads. We have seen and, in some cases, also helped a lot many enterprise businesses in banking, finance, energy, manufacturing, ISVs and other sectors run mission-critical production workloads on SQL Server on Linux and Containers.

SQL Server support for Linux began with the release of SQL Server 2017. The platforms supported at that time were Red Hat Enterprise Linux 7, SUSE Enterprise Linux Server v12, and Ubuntu 16.04 and 18.04. Initially, the SQL Server container images were based on the Ubuntu distribution only.

With the release of SQL Server 2022, support was extended to include the latest versions of the various Linux distributions, such as Red Hat Enterprise Linux 8, Ubuntu 20.04, and SUSE Enterprise Linux Server v15.

On August 11, 2023, we announced the availability of SQL Server 2022 for Red Hat Enterprise Linux 9 and Ubuntu 22.04 in preview mode. This preview release gives you the option to run SQL Server 2022 as a confined application when SELinux is enabled in enforcing mode. This

allows SQL Server to seamlessly integrate with a secure SELinux environment.

The new package called “mssql-server-selinux” enables the custom policies required to run SQL Server as a confined application with SELinux. You can also choose to continue running SQL Server as an unconfined application, as in previous versions of Red Hat Enterprise Linux. To do this, you can skip installing the new “mssql-server-selinux” package and just install the “mssql-server-package” as you did with previous Red Hat Linux versions.

To learn more about SELinux, you may refer to:

SQL Server on Linux - Architecture

If you are wondering how SQL Server was made to run on Linux, after a long history of only supporting Windows-based operating systems, then it is highly recommended you to read the complete blog: https://cloudblogs.microsoft.com/sqlserver/2016/12/16/sql-server-on-linux-how-introduction/ that was published on December 16, 2016 when the SQL Server 2017 was in public preview.

Here is a summary of the blog that explains how SQL Server on Linux really works. To support SQL Server on Linux, the most important task was to abstract the dependency of the SQL Server on Windows. SQL Server depends on various libraries and their functions. The semantics commonly used in Windows development fall basically into three categories, which were true for SQL Server as well:

Win32

NT kernel (ntdll.dll)

Windows application libraries (such as MSXML, CLR, and MSDTC)

So, how did we bring this abstraction of SQL Server from Windows? The SQL Platform Abstraction Layer(SQL PAL) was developed to bring this abstraction. The SQL PAL layer is a merger of SQL Server OS (SOS) and the user mode Library OS.

SQL OS (SOS) was introduced in SQL Server 2005 that allowed for a centralized set of low-level management and diagnostic functionality and minimized the number of system calls involved in scheduling execution by running non-preemptively and letting SQL Server do its own resource management. You can read more about SOS here: https://learn.microsoft.com/en-us/archive/blogs/slavao/platform-layer-for-sql-server#sqlos-architecture

The user mode library OS (LibOS) is a working Windows library that implements a subset of 1500+ Windows ABIs (Application Binary Interface), which means only about 45-50 ABIs related to memory management, host synchronization and I/O (network and disk) were needed to interact with the host. Also, it provides the capability of hosting other Windows components like CLR, MXSML and other APIs that SQL suites depend on.

Thus, the combination of SQL Server OS (SOS) and LibOS gave birth to SQL PAL, and the architecture shown in Figure 3.1 for reference is taken from the blog referred to above.

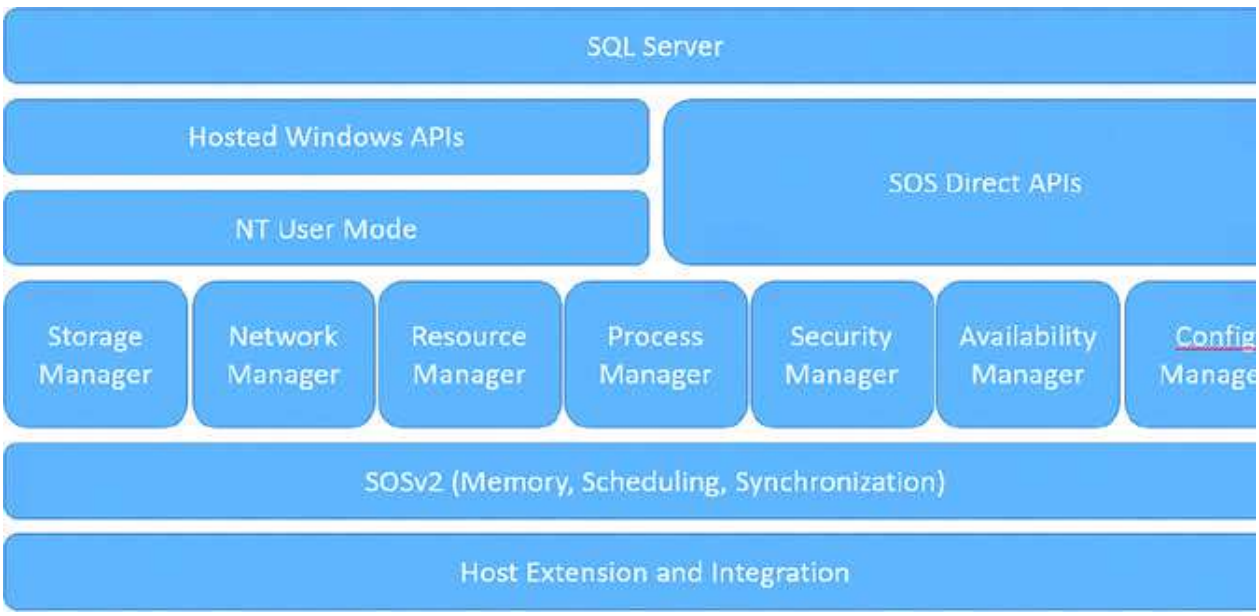


Figure 3.1: SQL Platform Abstraction Layer (PAL) Architecture

Important callouts of this architecture are:

You have a set of SOS direct APIs that don't go through the NTUM (NT user mode API or Win32) or the hosted Windows APIs like MSXML or MSDTC.

For the code that cannot be served by the SOS direct APIs that gets routed via the hosted Windows API or the NTUM.

But all the subsystems like storage, network or resource managed will be based on SOS and will be shared between SOS direct code and NTUM APIs accessed code.

Though this might look like a lot of layering, there are no real boundaries as such with most of the resource management happening within SQL PAL.

You might ask what the SQL Server Process looks like on Linux (refer to Figure which shows what the address space looks like when running you can learn more about this diagram from the blog post referred to earlier.

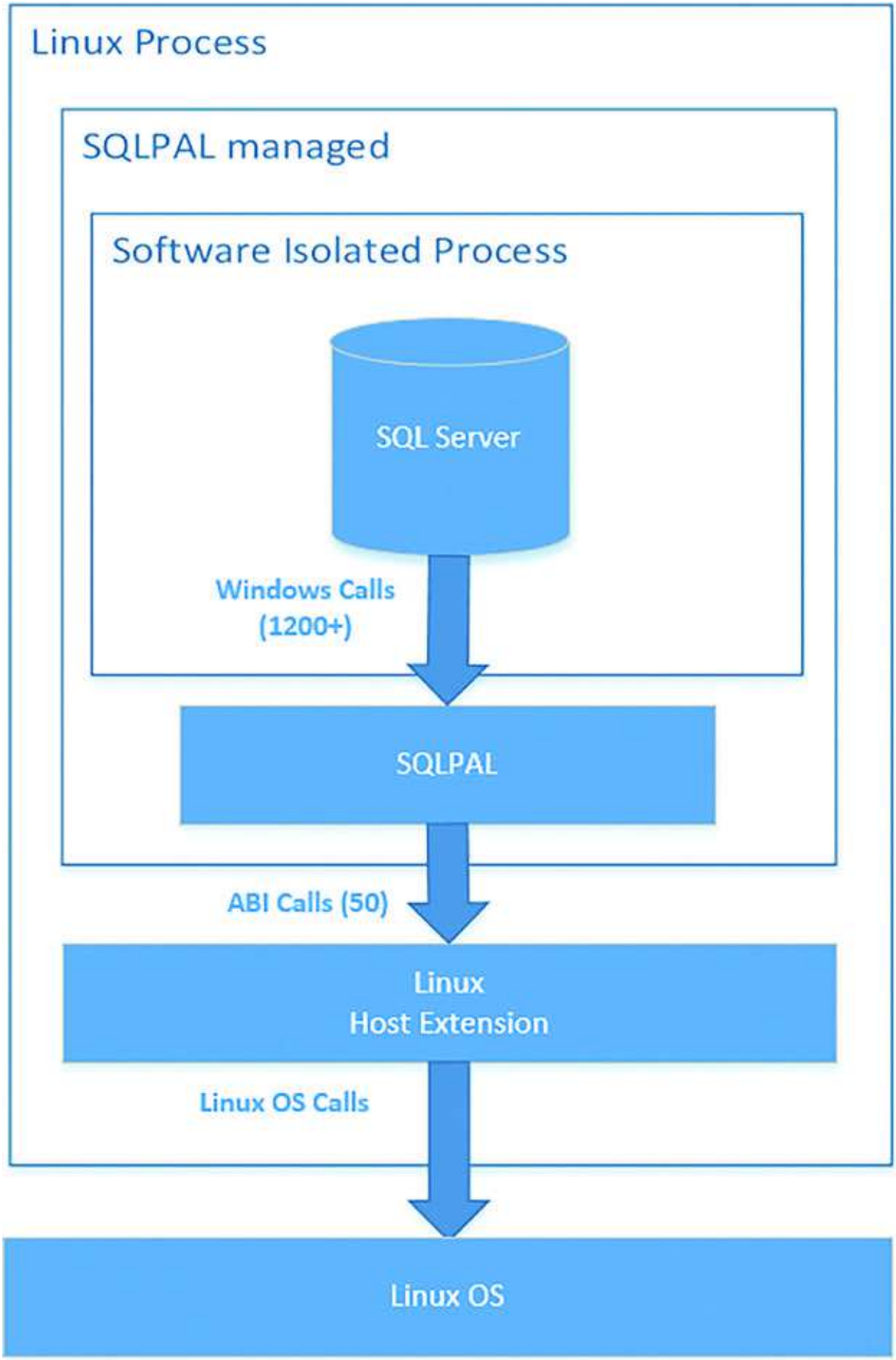

Linux Process
SQLPAL managed
Software Isolated Process
SQL Server
Windows Calls
(1200+)
SQLPAL
ABI Calls (50)
Linux
Host Extension
Linux OS Calls
Linux OS

Figure 3.2: SQL Server Linux Process

As soon as the process starts, the host extension is loaded which is simply a native Linux application.

Once the host extension starts, it loads and initializes SQLPAL, which then can launch software and isolated processes that are simply collection threads running within the same address space. This is how the sqldumper application is run to capture a crash dump when the SQL Server encounters a problem.

Finally, as soon as the SQLPAL is loaded and initialized, it then brings up SQL Server and voila you have the same SQL Server engine now up and running on Linux.

Now that we understand how SQL Server on Linux works, let us explore where and how to obtain Linux-based SQL Server packages.

Packaging and Availability of SQL Server on Linux and Containers for Production Workloads

The SQL Server Linux packages are released at the packages.microsoft.com repository The packages are arranged based on the distribution. You can find the SQL Server packages for Red Hat Enterprise Linux (RHEL), SUSE Linux Enterprise Server (SLES), and Ubuntu in the corresponding folders in the repository, as shown in Figure

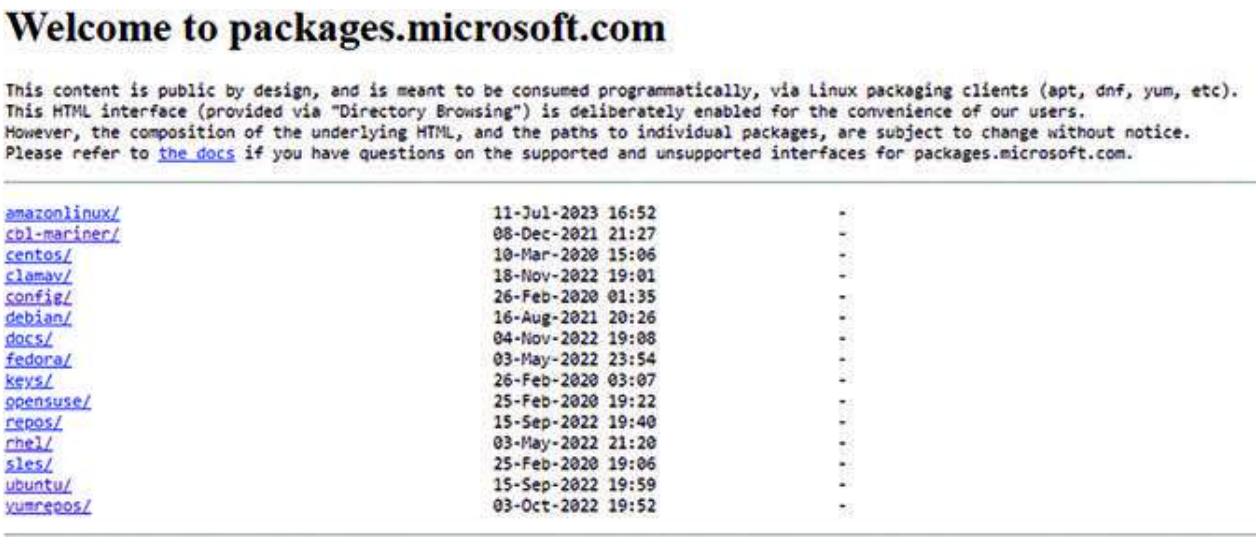


Figure 3.3: SQL Server packages repository

The SQL Server container images are published to the Microsoft Container Registry and are available in the following catalogs:

Internal Microsoft Artifact https://mcr.microsoft.com/en-us/catalog?search=sql%20server has SQL Server container images for both RHEL and Ubuntu-based images, as shown in Figure

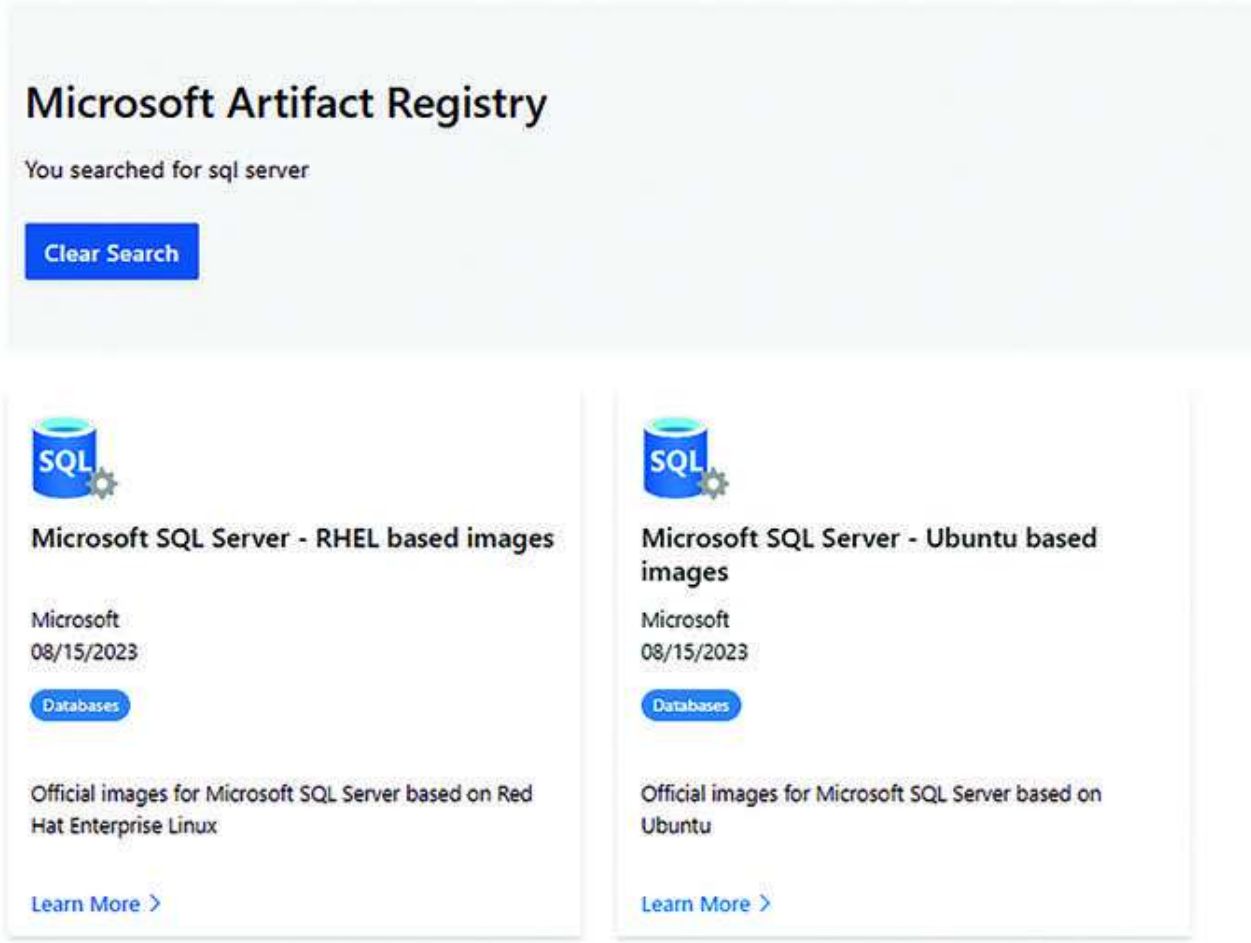


Figure 3.4: Microsoft Artifact Registry with SQL Server container images catalogede

On Docker This only enlists the SQL Server container images based on Ubuntu.

The Red Hat Ecosystem mssql/rhel/server - Certified Container Image - Red Hat Ecosystem Catalog. This enlists the SQL Server container images based on Red Hat UBI images.

You can use any of the catalogs to get started with SQL Server containers for production workloads. All these catalogs have the same SQL Server images, which are updated every time a cumulative update (CU) is released for the

SQL Server. The CU cadence for SQL Server on Linux is the same as for SQL Server on Windows. This means that whenever a CU is released for SQL Server on Windows, it is also released for SQL Server on Linux. The new CU-based SQL container image is released on the Microsoft Container Registry for both Red Hat UBI-based images and Ubuntu-based images. All the preceding catalogs are then updated to reflect the tags for the new CU.

Getting Started with SQL Server on Linux and Containers

To understand the packing and how you can get started with SQL Server on Linux for both servers, including VMs and containers, let us do a hands-on lab together. If you have installed SQL Server on Windows before, you will greatly appreciate the simplicity and the reduced time of SQL Server installation on Linux.

For this hands-on lab, you will need just one pre-requisite, that is:

A Microsoft Azure account, to be able to create the Virtual Machine. If you can already create a virtual machine anywhere else that is connected to the internet, then you don't need the Azure account for this demo.

Let us go ahead and create a Virtual machine on Azure that is going to run Red Hat Enterprise Linux version 8. We could also have chosen Ubuntu or SLES-based distributions for this demo. We are using Red Hat Linux to show how later in the demo we can automate the SQL Server installation using scripting tools like Ansible.

Figure 3.5 illustrates the VM creation blade from the Azure portal. We provide details like the region, and the image that we are using to create the VM in Red Hat Enterprise Linux 8.4. Also, we are creating the user amvin on the machine, and this is a small VM with 4 CPUs and 16 GB of memory. This size of the VM is good enough to showcase the installation of SQL Server.

For this demo, we are not adding any extra disks but as you will see in the next chapters where we discuss best practices for deploying SQL Server on Azure-based VMs, it is recommended to deploy extra disks to be used as data disk with the right RAID and filesystem configurations. We will talk about these details in the later chapters of this book.

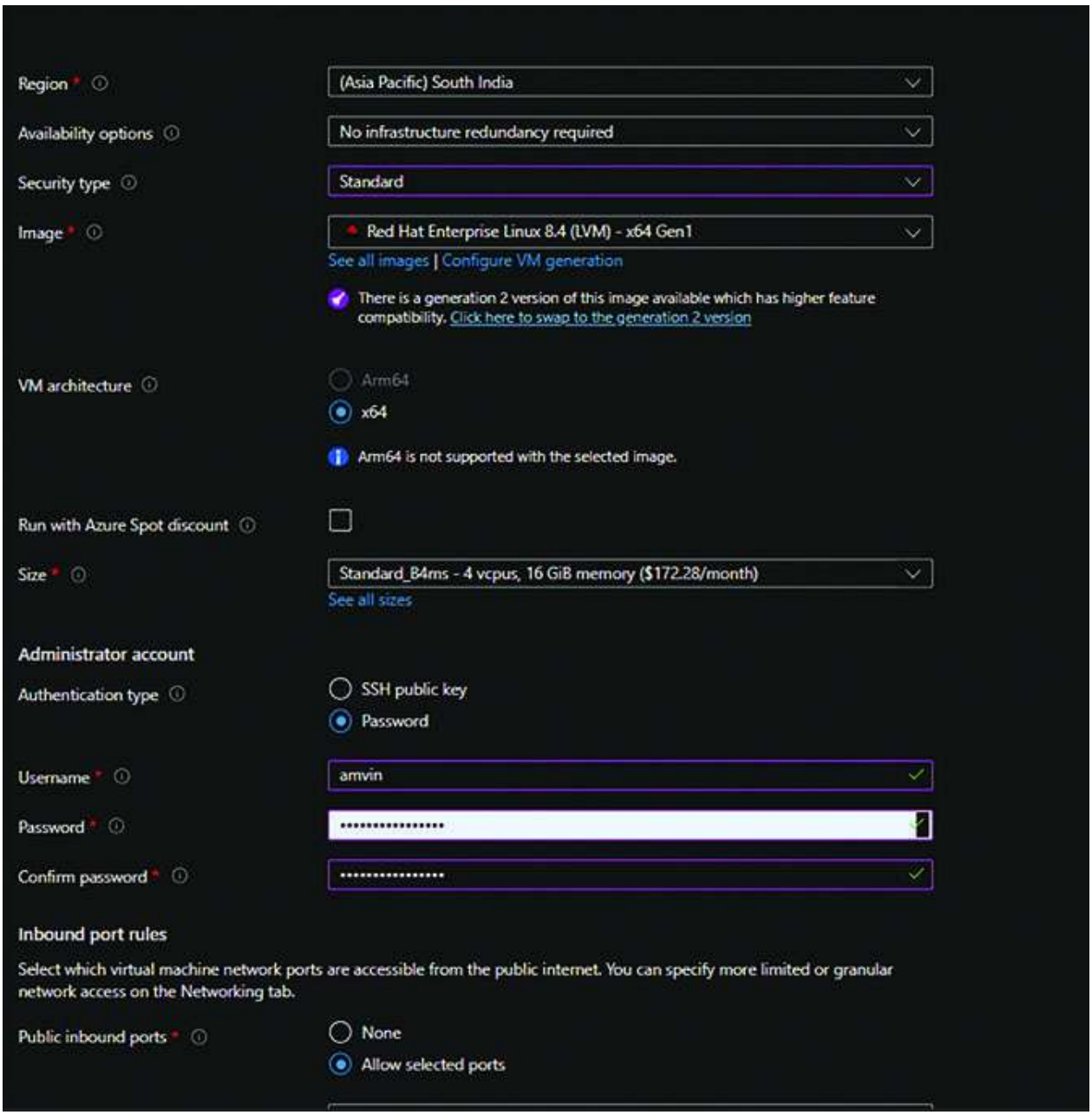


Figure 3.5: Creating Linux-based VM through the Azure Portal

For further details, refer to the hands-on lab for this chapter on the GitHub repository for this book.

Once you complete providing the details for the Networking, Management, and Monitoring sections of the VM, you can go with the default options, click the create button and once the VM is ready, you can log into the Linux VM using your choice of tools to SSH into the Linux VMs. It is suggested to use MobaXterm utility as it provides with an option to log into Windows and Linux VMs using the same utility.

Alternatively, Visual Studio Code (VS Code) can be used to establish a terminal connection to your Linux VM. To do this, open VS Code, install the Remote Explorer extension, and input your SSH details. For example, to connect to a Linux VM named rhel9sql with the IP address 10.0.0.117 using the username amvin, you would use the command: ssh After entering the user's password, you will be able to access the terminal as shown in Figure 3.6

Figure 3.6 (a): Creating Linux-based VM through the Azure Portal

From the MobaXterm, we connect to the newly created Azure RHEL-based VM as shown in Figure 3.6 We need to type in the password for the first time and then the terminal remembers it for easy login.

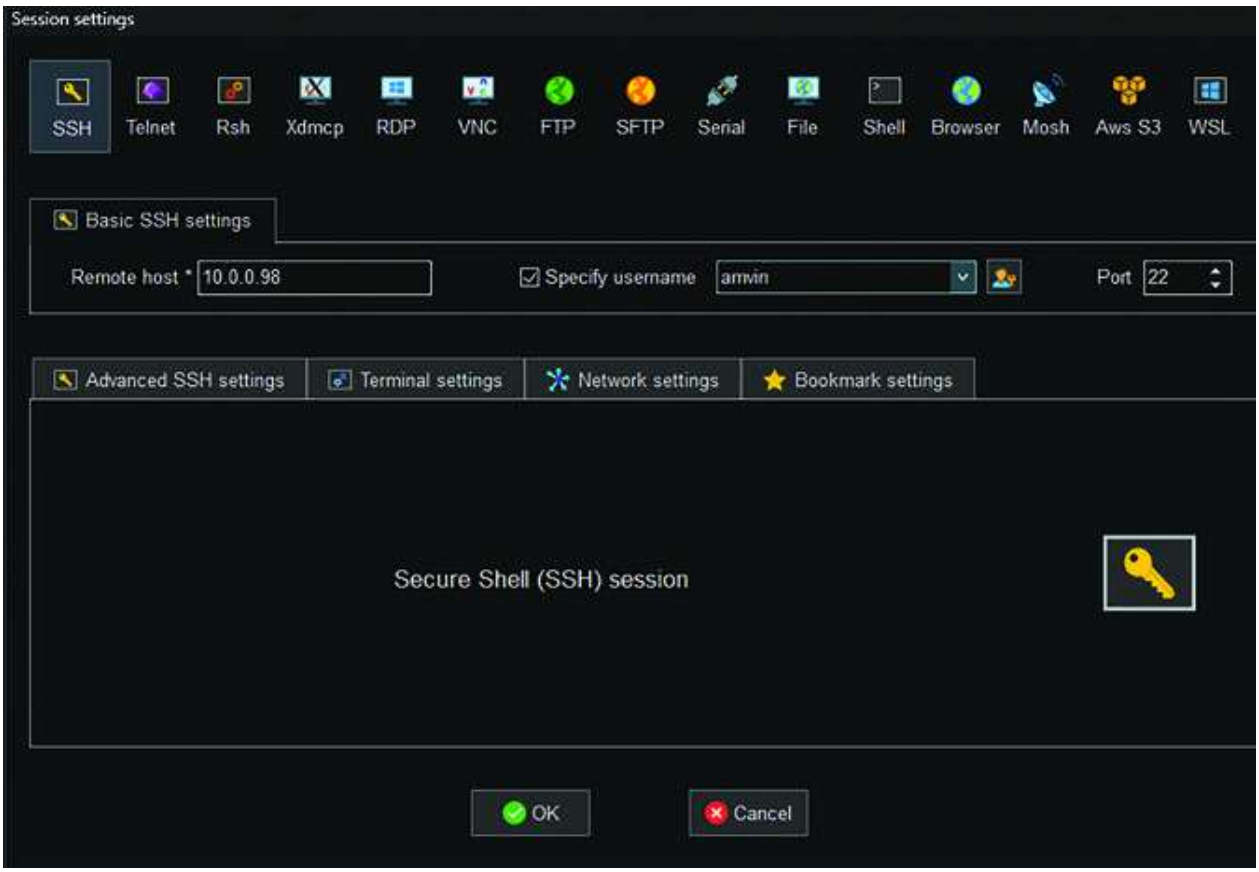


Figure 3.6 (b): MobaXterm screen to connect to Linux-based VMs

SQL Server Installation

The installation of the SQL Server and tools on Red Hat Linux is a simple five-step process including the configuration of the SQL Server.

Step 1:

Download the required repository configuration files that have the details to install SQL Server and tools. The commands are:

sudo curl -o /etc/yum.repos.d/mssql-server.repo https://packages.microsoft.com/config/rhel/8/mssql-server-2022.repo

To download the tools repository, change to super user mode with the following command:

Sudo su

Now download the repository for SQL tools using the command:

curl https://packages.microsoft.com/config/rhel/8/prod.repo > /etc/yum.repos.d/mssql-release.repo

exit super use mode with the command: exit

Step 2:

With the repositories configured, it’s time to run the installation. Since we are going to install the latest SQL Server 2022 package, we run the following commands with the version details.

The command below installs the SQL Server package. For specific packages, you can #modify the command as sudo yum install -y mssql-server-.x86_64 #Example:
#If you intend to install SQL Server 2022 CU 3 and not the latest SQL 2022 CU 7, then #change your install command to sudo yum install mssql-server-16.0.4025.1-13.x86_64

sudo yum install -y mssql-server
sudo yum install -y mssql-tools18

Step 3:

Once the installation is successful, you can go ahead and configure SQL Server using the mssql-conf tool, using this tool you select the edition of SQL Server, the sa login password, and accept the required EULAs necessary for the installation and use of SQL Server.

sudo /opt/mssql/bin/mssql-conf setup

Step 4:

Open any firewall ports needed to allow connections to the SQL Server port. In our case, we have not changed the default port:

sudo firewall-cmd --zone=public --add-port=1433/tcp --permanent
sudo firewall-cmd –reload

Step 5:

Ensure the SQL Server service is running, and you are ready to connect with the following commands:

systemctl status mssql-server

Here is how the entire setup would look like, when we run this on a Linux VM. For reference the Figure 3.7 (a) and Figure 3.7 (b) shows you how the installation of SQL Server looks on a Linux VM:

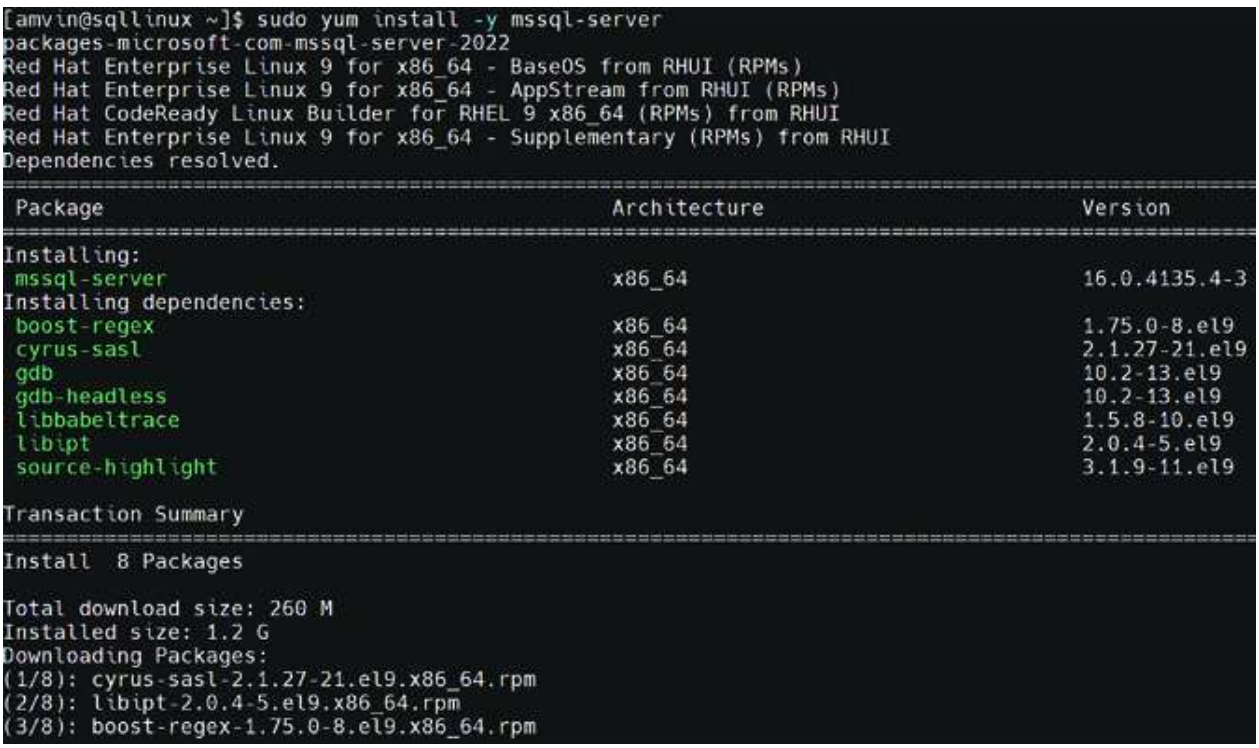


Figure 3.7 (a): Partial Screenshot of the SQL Server installation

In Figure 3.7 you will notice that the setup screen suggests you to run the configuration of the SQL Server after completion of the installation of binaries.

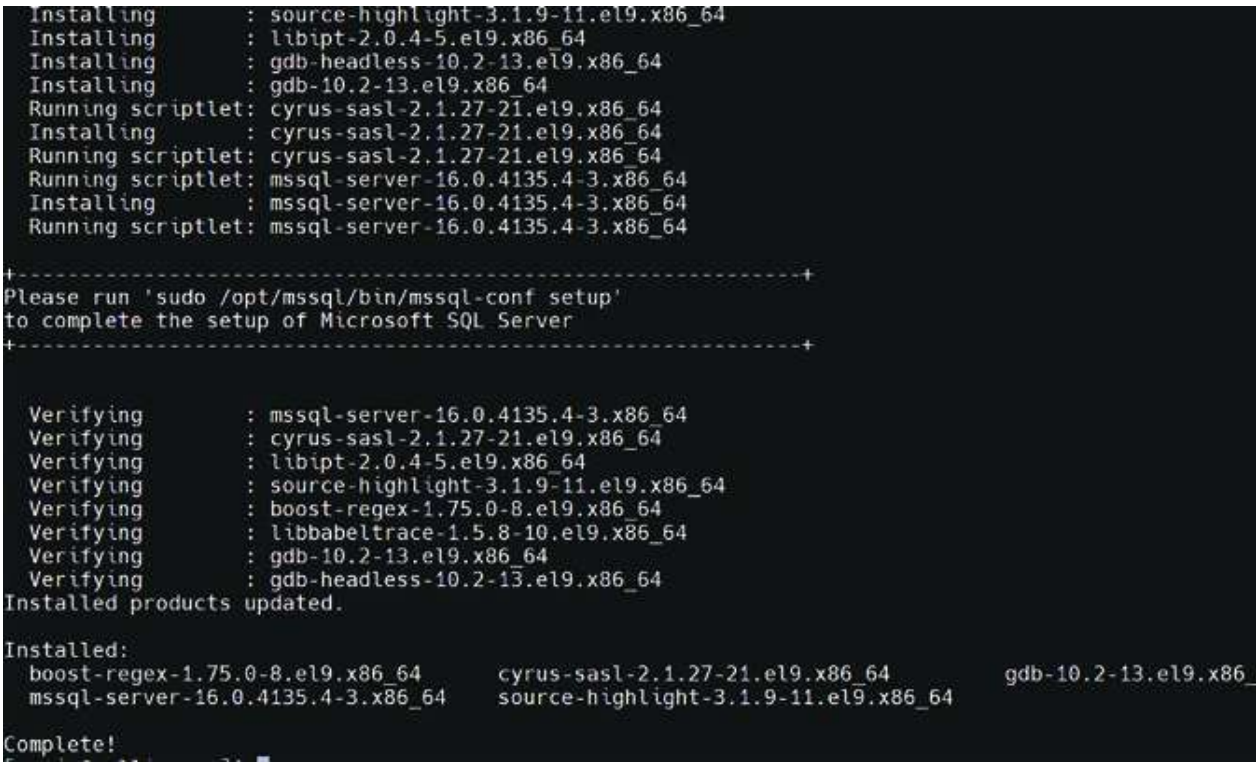


Figure 3.7 (b): Completion of SQL Server installation and the suggestion to run the configuration of SQL Server

Once the SQL Server installation is finished, it's recommended to proceed with the configuration. During this step, you will select the desired SQL Server edition, accept the End User License Agreement (EULA), and set the SA account password, which will be used to connect to SQL Server after the configuration is complete and the service starts. Figure 3.7 (c) provides an overview of these processes.

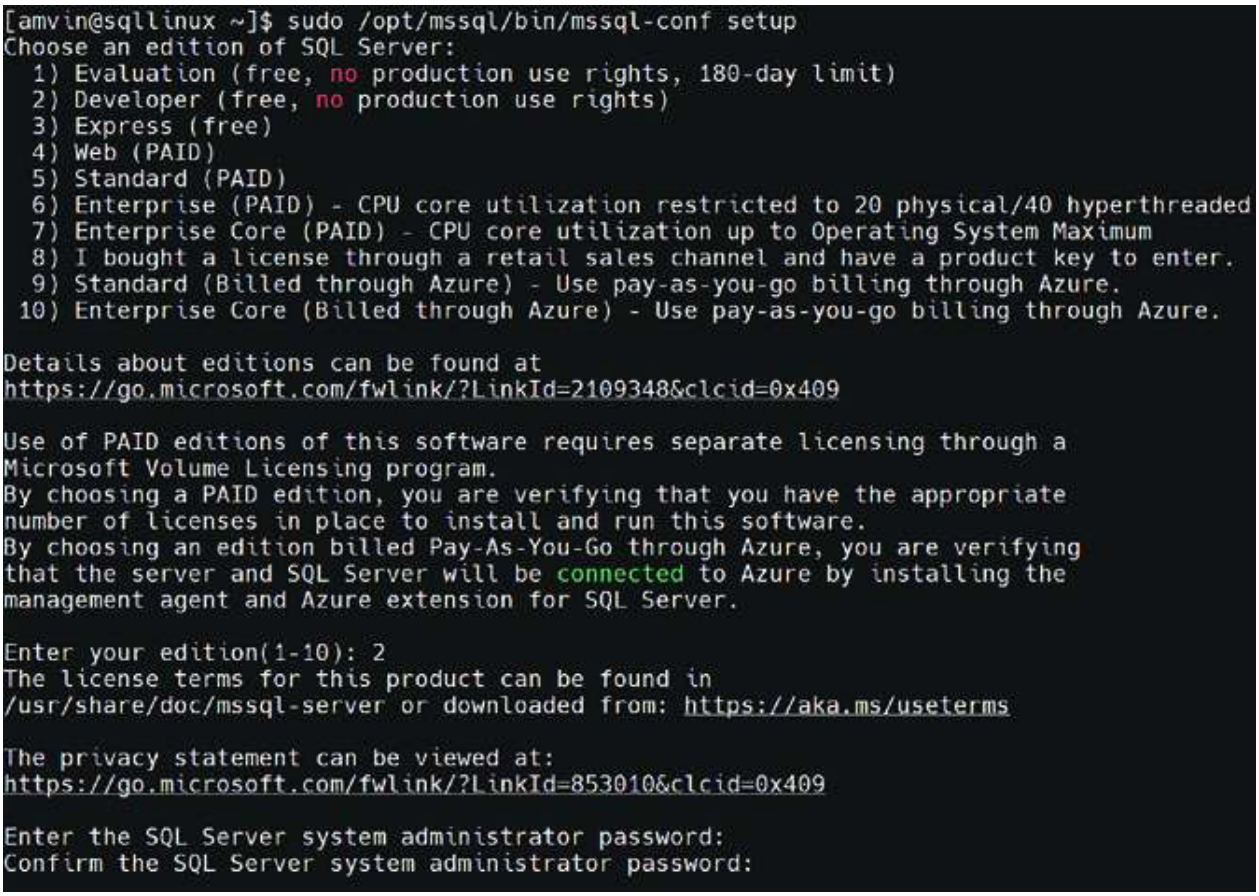


Figure 3.7 (c): Configuring the SQL Server instance using the mssql-conf setup

SQL Server Management Studio (SSMS) or Azure Data Studio (ADS) can be used to connect to the SQL Server on Linux instances, and it works the same way as you would work with SQL Server on Windows. Figure 3.7 (d) shows an illustration of SSMS connected to SQL Server on a Linux instance.

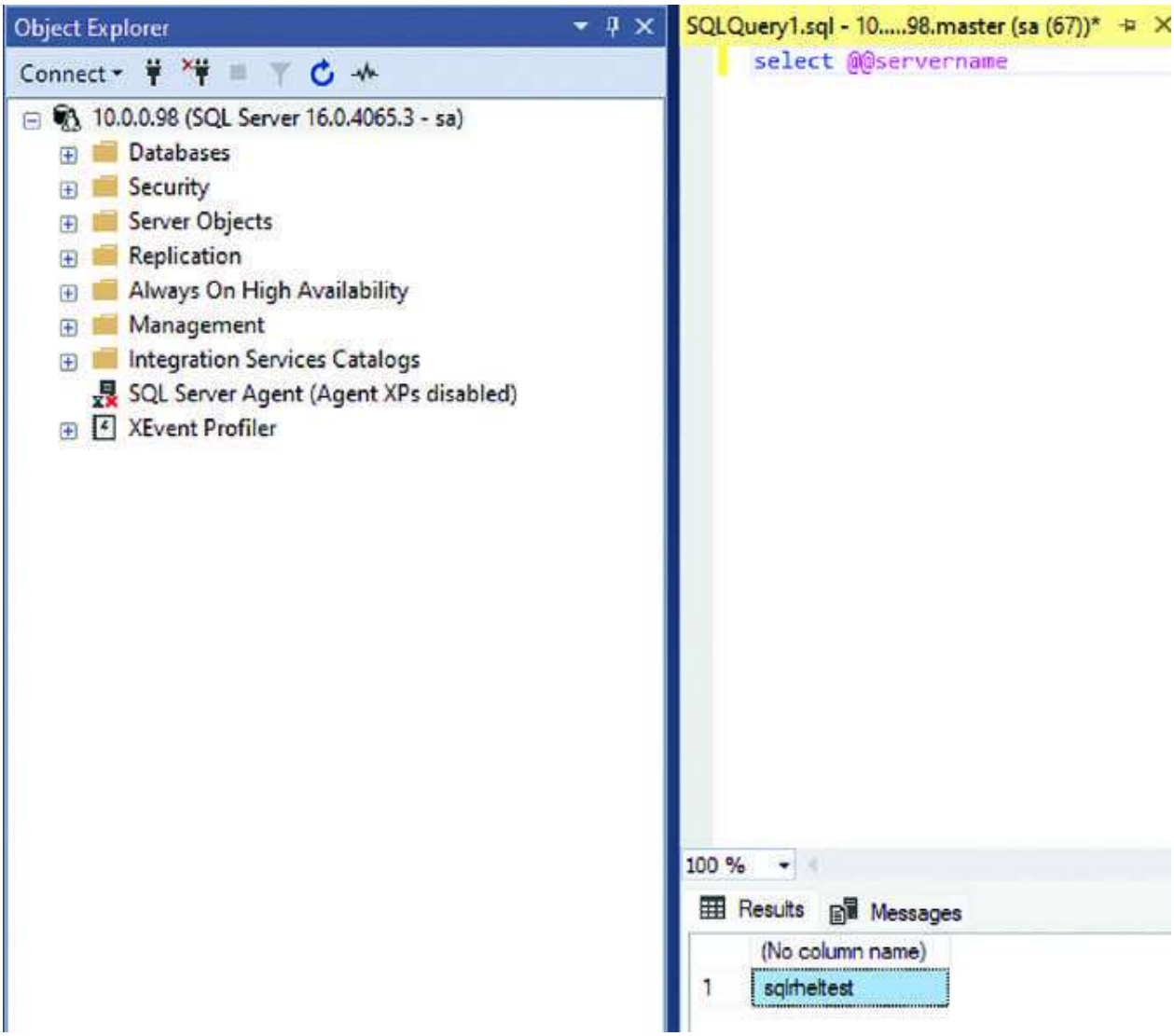


Figure 3.7 (d): Connecting to SQL Server on Linux instance using SSMS

SQL Server Installation on Multiple Targets using Ansible

You can automate the installation of SQL Server across multiple targets running Red Hat Linux-based machines using the Ansible scripts. We worked closely with the Red Hat Engineering team who developed the "ansible-collection-microsoft-sql" that you can use to deploy SQL Server on a scale. This Ansible system role is available with Red Hat Enterprise Linux deployments by default and can be installed with a simple command:

"sudo yum install ansible-collection-microsoft-sql"

The way Ansible-based installation works is that you have a central controller node that has passwordless ssh access configured for all the targets, as shown in Figure These are called the target servers where you intend to install SQL Server. You need to install the Ansible core, and the Ansible SQL system role: ansible-collection-microsoft-sql on the controller node only.

The controller node has a host file that has the target server IP address mentioned. This helps the controller know the targets it is going to connect to and install the SQL server on. Once you are ready for the installation to happen, create the playbook.yaml file with the required environment variables defining the configuration for the SQL Server. Once ready, run one command that goes ahead and installs the SQL Server across the targets.

ansible-playbook -u root playbook.yaml

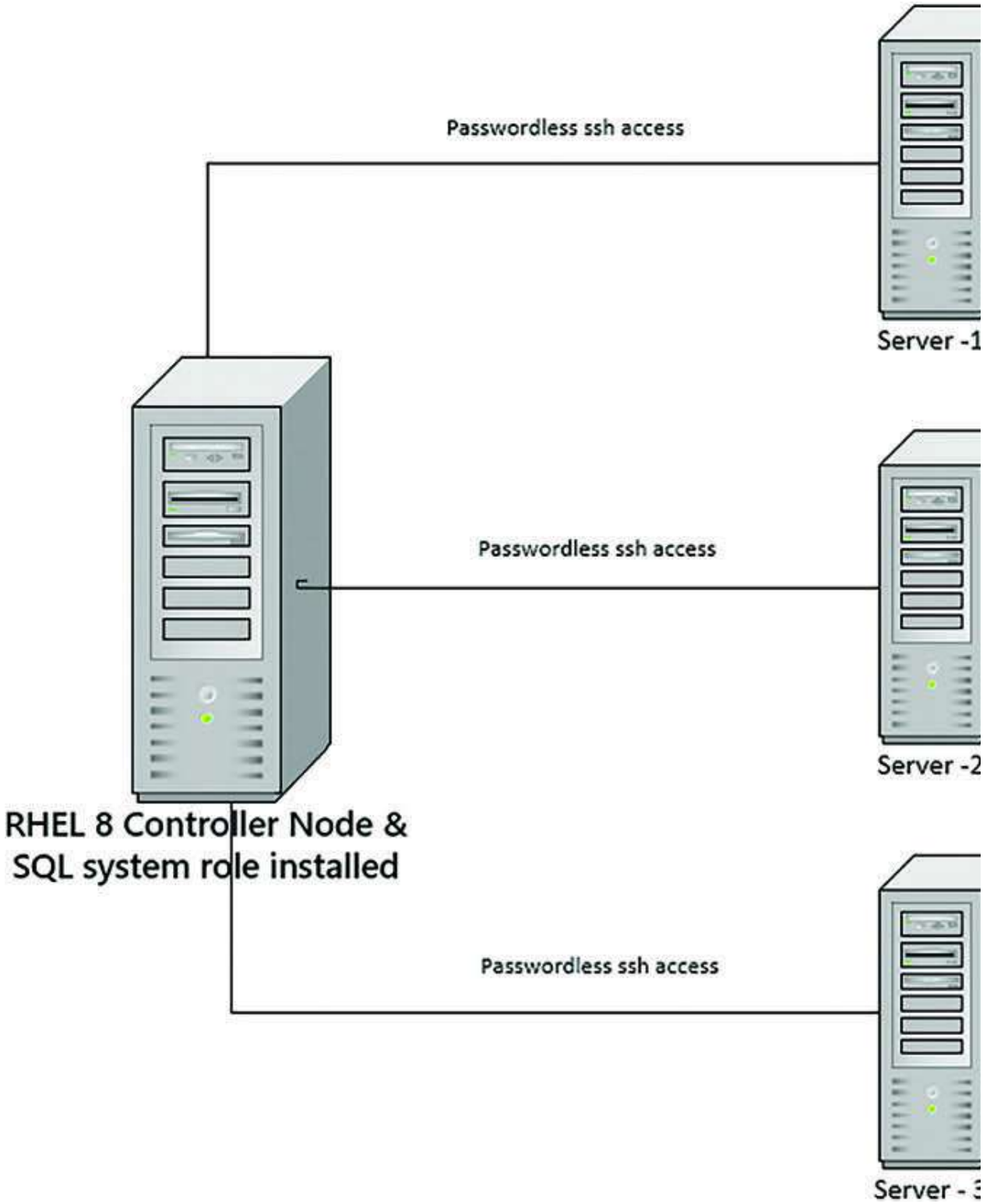


Figure 3.8: Ansible-based Controller node with multiple targets to install and configure SQL Server

To understand how you can install the ansible-core, the ansible system role, create the host file entries for the target servers and finally create the

Deploy SQL Server – The Ansible way! - Microsoft Community Hub

As you might have already realized, there is no SQL Server Configuration Manager UI, as you do for SQL Server on Windows. For SQL Server on Linux deployments, all of your configurations are done through the All the changes you make using the mssql-conf tool get saved in the mssql.conf file that is available at /var/opt/mssql/ location. Here are a few examples of performing common tasks using the mssql-conf tool.

```
#To enable SQL Server agent run:
sudo /opt/mssql/bin/mssql-conf set sqlagent.enabled true

#to set a trace flag in SQL Server:
sudo /opt/mssql/bin/mssql-conf traceflag 3979 on
#to turn off a trace flag for SQL Server
sudo /opt/mssql/bin/mssql-conf traceflag 3979 off

# change the TCP port for SQL Server from 1433 to 19871
sudo /opt/mssql/bin/mssql-conf set network.tcpport 19871

#restart SQL Server using the command:
sudo systemctl restart mssql-server
```

Please note for SQL Server on Linux, the account under which the SQL Server runs is the mssql user. The binaries are placed at You cannot change the user or the location of the binaries as of writing this book.

Getting Started with SQL Server Containers

As discussed earlier in this chapter, the SQL Server container images are available based on two different base OS images and they are:

SQL Server container images based on Ubuntu-based OS image

SQL Server container images based on RHEL-based images

It is important that when you are running SQL Server container images for production workload, please ensure that the host Linux OS version matches the container base OS image. For example, if you are running the SQL Server container- Ubuntu-based images on Ubuntu-based worker nodes running Kubernetes service, or an Ubuntu-based Standalone host machine. You should not mix and match the container base OS image and the host OS distributions. If you want to run production workloads for SQL Server containers on RedHat OpenShift, then you should use the SQL Server container images built on RHEL-based OS images and not the SQL container images built on Ubuntu-based images.

Customizing the SQL Server container images is also supported. Here is an example of a docker file that is available here: mssql-docker/linux/preview/SLES at master · microsoft/mssql-docker (github.com) that can be used to create SQL Server container images based on SUSE Enterprise Linux.

#

```
# Note: This image requires an active SLES15 subscription to build.
#
# Thank you to our SUSE Partners for helping with this :)
#
# Assumptions
# 1. use a matching version to the underlying build host
# 2. ensure it is registered to have access to needed repos
# then leveraging container-suseconnect-zypp
# e.g. zypper ref
# Repository ‘SLE-Module-Containers12-Pool’ is up to date.
# Repository ‘SLE-Module-Containers12-Updates’ is up to date.
# Repository ‘SLES12-SP5-Pool’ is up to date.
# Repository ‘SLES12-SP5-Updates’ is up to date.
# All repositories have been refreshed.
# 3. minimize the layers by consolidating commands

FROM registry.suse.com/suse/sle15:15.3

ENV ADDITIONAL_MODULES=sle-module-legacy

RUN zypper install --no-confirm --no-recommends \
# install setcap to be used later
# curl is needed for rpm import
libcap-progs curl && \
rpm --import https://packages.microsoft.com/keys/microsoft.asc && \
zypper rm --no-confirm --clean-deps curl

# consider merging the two RUNs to save ~ 40mb at the cost of caching
adding the signing key
# add mssql-server repo
```

```
RUN zypper addrepo --no-check
https://packages.microsoft.com/config/sles/15/mssql-server-2019.repo && \
zypper refresh packages-microsoft-com-mssql-server-2019 && \
# install mssql-server
zypper install --no-confirm --auto-agree-with-licenses --no-recommends
mssql-server && \
# add mssql-tools repo
zypper addrepo --check
https://packages.microsoft.com/config/sles/15/prod.repo && \
zypper refresh packages-microsoft-com-prod && \
# install mssql-tools (consider removing to reduce size) Microsoft already
maintains a separate mssql-tools image
ACCEPT_EULA=Y zypper install --no-confirm --no-recommends mssql-
tools && \
zypper clean --all && \
# post installation of SQL Server the mssql user/group is created
# so set the right permissions to the msssql folder
mkdir -p -m 770 /var/opt/mssql && \
chown -R mssql /var/opt/mssql && \
# grant sql the permissions to connect to ports <1024 as a non-root user
setcap ‘cap_net_bind_service+ep’ /opt/mssql/bin/sqlservr && \
# allow dumps from the non-root process
setcap ‘cap_sys_ptrace+ep’ /opt/mssql/bin/paldumper && \
setcap ‘cap_sys_ptrace+ep’ /usr/bin/gdb && \
# ldconfig file because setcap causes the os to remove LD_LIBRARY_PATH

# and other env variables that control dynamic linking
mkdir -p /etc/ld.so.conf.d && \
touch /etc/ld.so.conf.d/mssql.conf && \
echo -e “# mssql libs\n/opt/mssql/lib” >> /etc/ld.so.conf.d/mssql.conf && \
ldconfig
```

EXPOSE 1433

USER mssql

CMD [“/opt/mssql/bin/sqlservr”]

Similarly, you can add other SQL Server components like the Agent, Full-text search, Polybase and more and create a customized SQL Server image. For more examples you may refer to the official Microsoft mssql-docker repository on GitHub: mssql-docker/linux/preview/examples at master · microsoft/mssql-docker (github.com)

Getting started with the container images is as easy as installing the docker engine 1.8+ or any other container runtime on the stand-alone host and then pulling the container image from the Microsoft container repository and then running a container using that image. Here is a quick example to help you understand.

docker pull mcr.microsoft.com/mssql/server:2022-latest

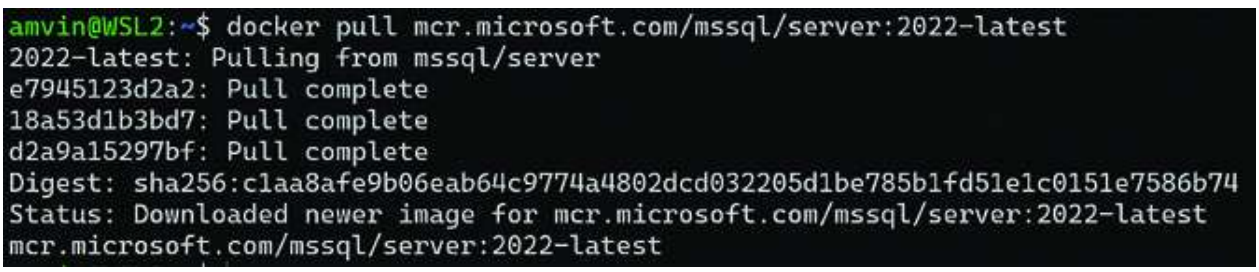


Figure 3.9 (a): Pulling SQL Server container image using docker

Once the image is pulled, you can run the container using the command:

docker run -e “ACCEPT_EULA=Y” -e “MSSQL_SA_PASSWORD=yourStrong(!)Password” -e “MSSQL_PID=Evaluation” -p 1433:1433 --name sqlpreview --hostname sqlpreview -d mcr.microsoft.com/mssql/server:2022-preview-ubuntu-22.04

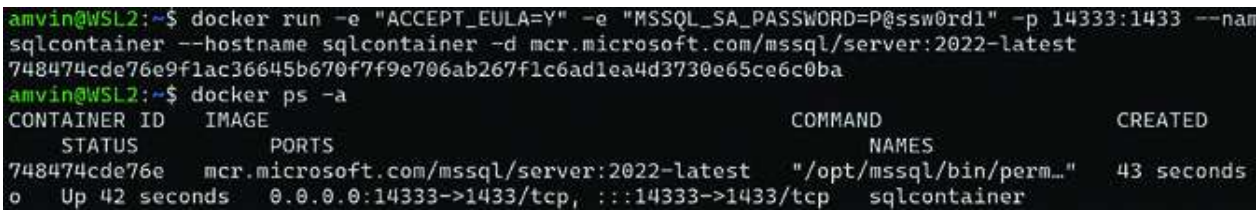


Figure 3.9 (b): Deploying SQL Server container image and listing the containers running using docker command

In the preceding command, you can notice that we pass the mssql-conf settings as environment variables, examples are the EULA acceptance using the variable then the SA password is passed using the MSSQL_SA_PASSWORD variable, followed by the server name or the hostname is set to sqlpreview and the port on the host where the SQL Server container instance is listening to is mapped to 1433 on the host, you can always change this to your choice of port.

Then, you can connect to the SQL Server container running on a standalone host, you will use the host IP address and the port that you mapped in the docker run command to connect to the SQL Server using tools like SSMS and ADS.

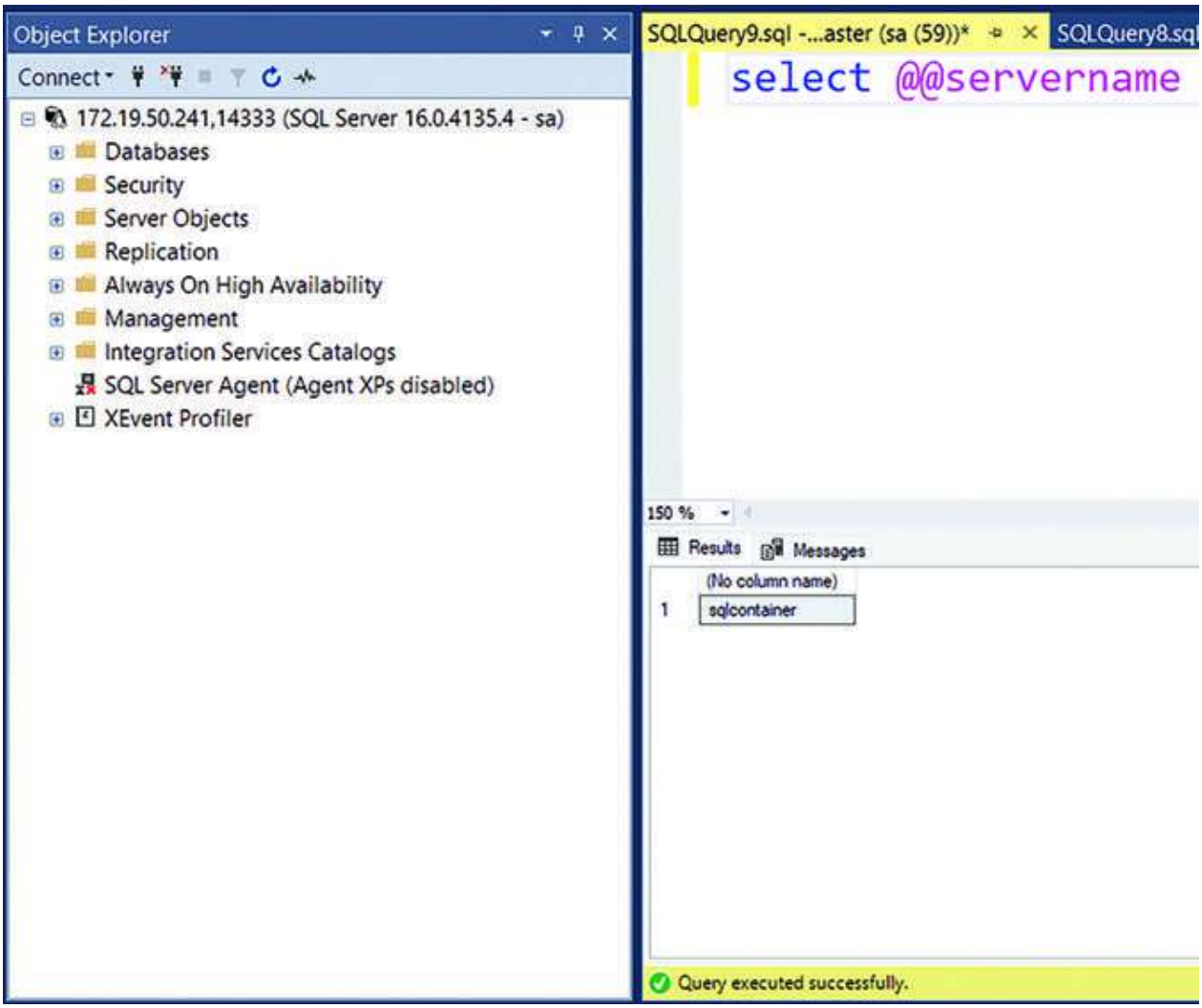


Figure 3.9 Connecting to SQL Server container using SSMS

SQL Server Deployed on the Kubernetes Platform

To deploy SQL Server containers on Kubernetes, you can use the deployment scripts that deploy the required objects including the SQL Server pods on the Kubernetes cluster. You will need a Kubernetes cluster, such as the Azure Kubernetes Service (AKS) cluster that we used in previous examples. Once you have a Kubernetes cluster, you can connect to it using kubectl from any client machine, regardless of the operating system.

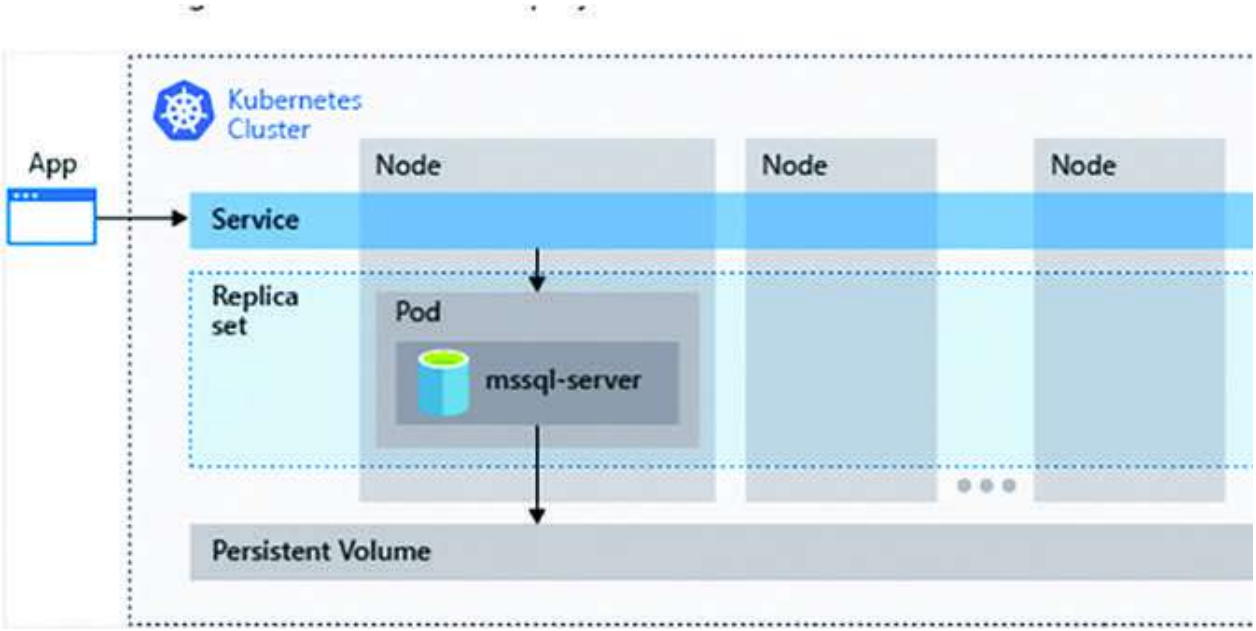


Figure 3.9(d): SQL Server container deployment on the Kubernetes platform

Continuing to use the Kubernetes cluster we created previously, we are going to deploy the SQL Server Pod on the Kubernetes cluster using the following script.

From my Windows client machine, we open the command prompt and use the kubectl tool that we installed previously, to connect to the Azure Kubernetes cluster testk8s using the command:

```
az aks get-credentials --resource-group azurearc_test --name testk8s
```

Then we go ahead and create the secret object on the Kubernetes cluster to store the SA password that will be passed when deploying the SQL Server pods on the container.

```
kubectl create secret generic mssql --from-literal=MSSQL_SA_PASSWORD=”MyC0m9l&xP@ssw0rd”
```

This is the deployment script that we use which will go ahead and deploy one SQL Server pod that runs one single SQL Server container inside the pod. Also, the script creates an external load balancer service that enables connection to the SQL Server instance deployed inside the Kubernetes cluster. Create a file with any name and save it as a .yaml file. In this case, we are creating a file called sqldeploy.yaml and the content of the file is as follows. For details on each of these parameters, refer to the GitHub repository for this chapter. (SQL-Server-Unveiled---Path-to-Data-Excellence/Chapter 3 at main · ava-orange-education/SQL-Server-Unveiled---Path-to-Data-Excellence (github.com)

```
kind: StorageClass
apiVersion: storage.k8s.io/v1
metadata:
name: azure-disk
provisioner: kubernetes.io/azure-disk
parameters:
storageaccounttype: Standard_LRS
```

```
kind: Managed
---
apiVersion: apps/v1
kind: StatefulSet
metadata:
name: mssql
labels:

app: mssql
spec:
serviceName: “mssql”
replicas: 1
selector:
matchLabels:
app: mssql
template:
metadata:
labels:
app: mssql
spec:
securityContext:
fsGroup: 10001
containers:
- name: mssql
image: mcr.microsoft.com/mssql/server:2022-latest
ports:
- containerPort: 1433
name: tcpsql
env:
- name: ACCEPT_EULA
value: “Y”
- name: MSSQL_ENABLE_HADR
value: “1”
```

- name: MSSQL_AGENT_ENABLED
value: “1”
- name: MSSQL_SA_PASSWORD
valueFrom:

secretKeyRef:
name: mssql
key: MSSQL_SA_PASSWORD
volumeMounts:
- name: mssql
mountPath: “/var/opt/mssql”
volumeClaimTemplates:
- metadata:
name: mssql
spec:
accessModes:
- ReadWriteOnce
resources:
requests:
storage: 8Gi
---
apiVersion: v1
kind: Service
metadata:
name: mssql-0
spec:
type: LoadBalancer
selector:
statefulset.kubernetes.io/pod-name: mssql-0
ports:
- protocol: TCP
port: 1433
targetPort: 1433

name: tcpsql

Once, you have saved the sqldeploy.yaml file, you can run the script using the following command:

kubectl apply -f "D:\PM Stuff\k8s setup\Statefulset-deployment\ sqldeploy.yaml"

Wait for a few minutes and you should have the SQL Server pod running, as shown in Figure

Figure 3.10: Viewing SQL Server container deployments on Kubernetes

You can connect using the SSMS tool and query the SQL Server instance, as shown in Figure

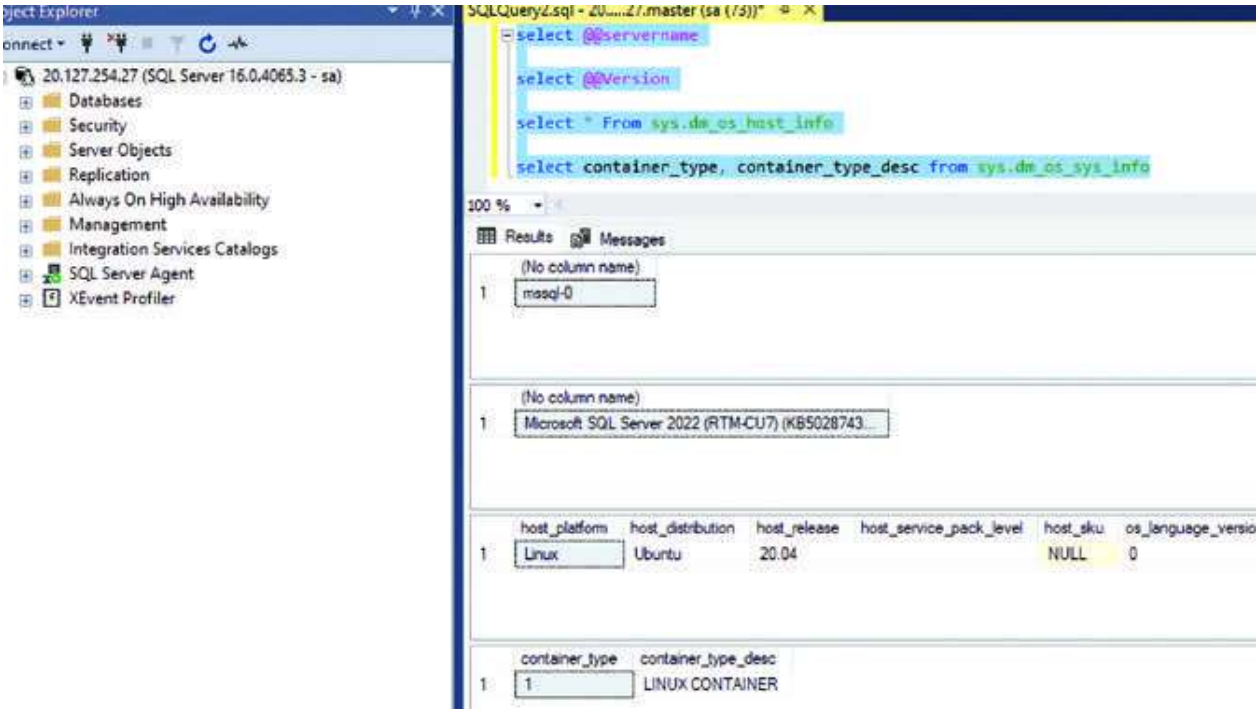


Figure 3.11: Connecting to SQL Server containers via SSMS

Azure SQL

In the previous section of this chapter, we looked at the different points that you should consider before narrowing down your choices and selecting the SQL offering that suits you the best. Once you've chosen to use a Cloud-based SQL offering, let us discuss the various options that are available under the Azure SQL family.

The scope of this chapter, when talking about SQL offerings in the Cloud, is specific to Microsoft Azure-based SQL cloud offerings. The suite of Microsoft SQL offerings available on Azure is called "Azure SQL", this consists of SQL Server on Azure Virtual Machines (VMs), Azure SQL Database (Azure SQL DB), Azure SQL Managed Instance and Azure SQL Edge. Again, let us visualize the choices for a better understanding, as depicted in Figure

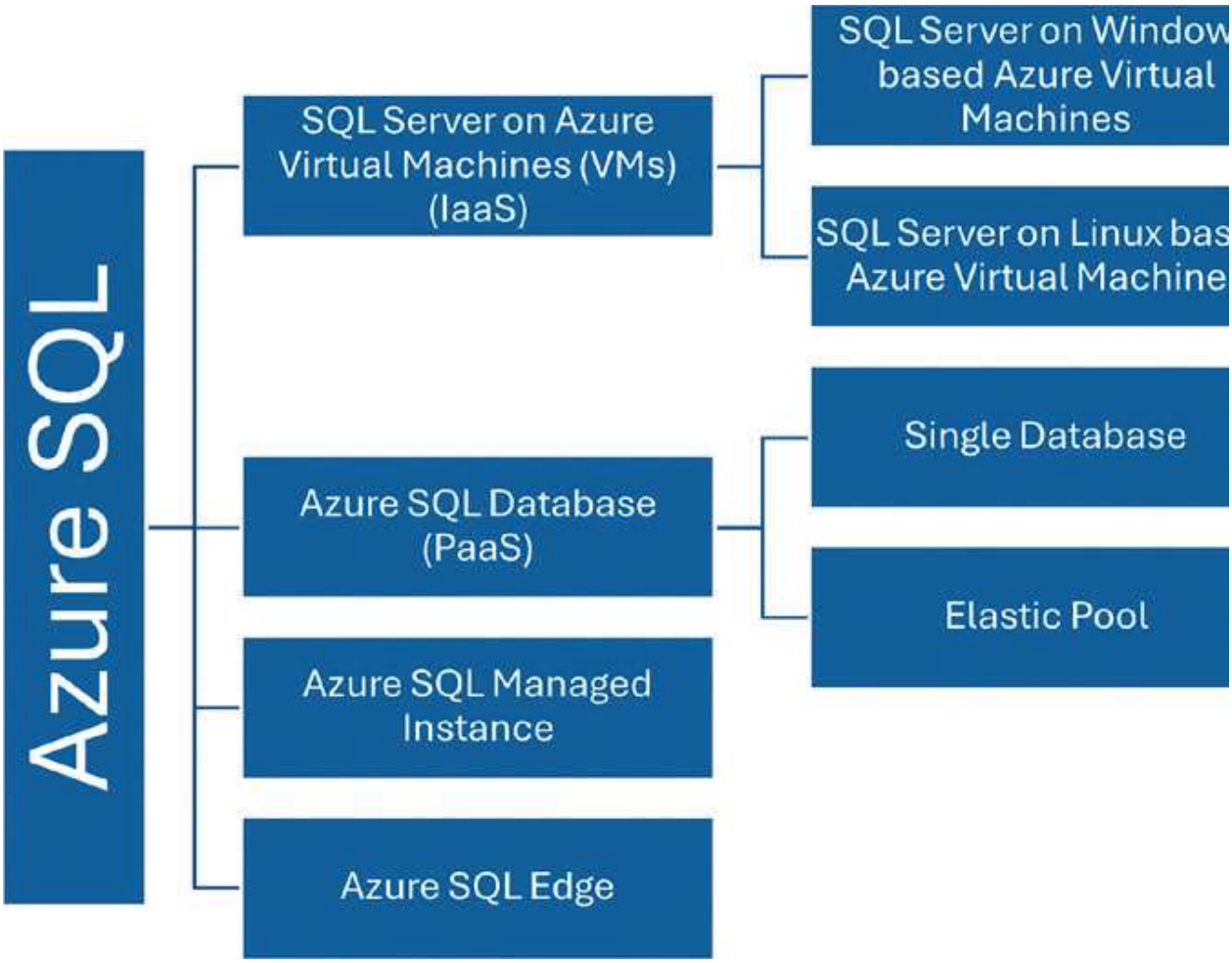


Figure 3.12: Azure SQL family

Figure 3.13 is a diagram that's available on the official Microsoft documentation: What is Azure SQL? - Azure SQL | Microsoft Learn which is informative as it helps explain the cost versus the level of administration you get over the infrastructure for each service offering including the on-premises deployments.

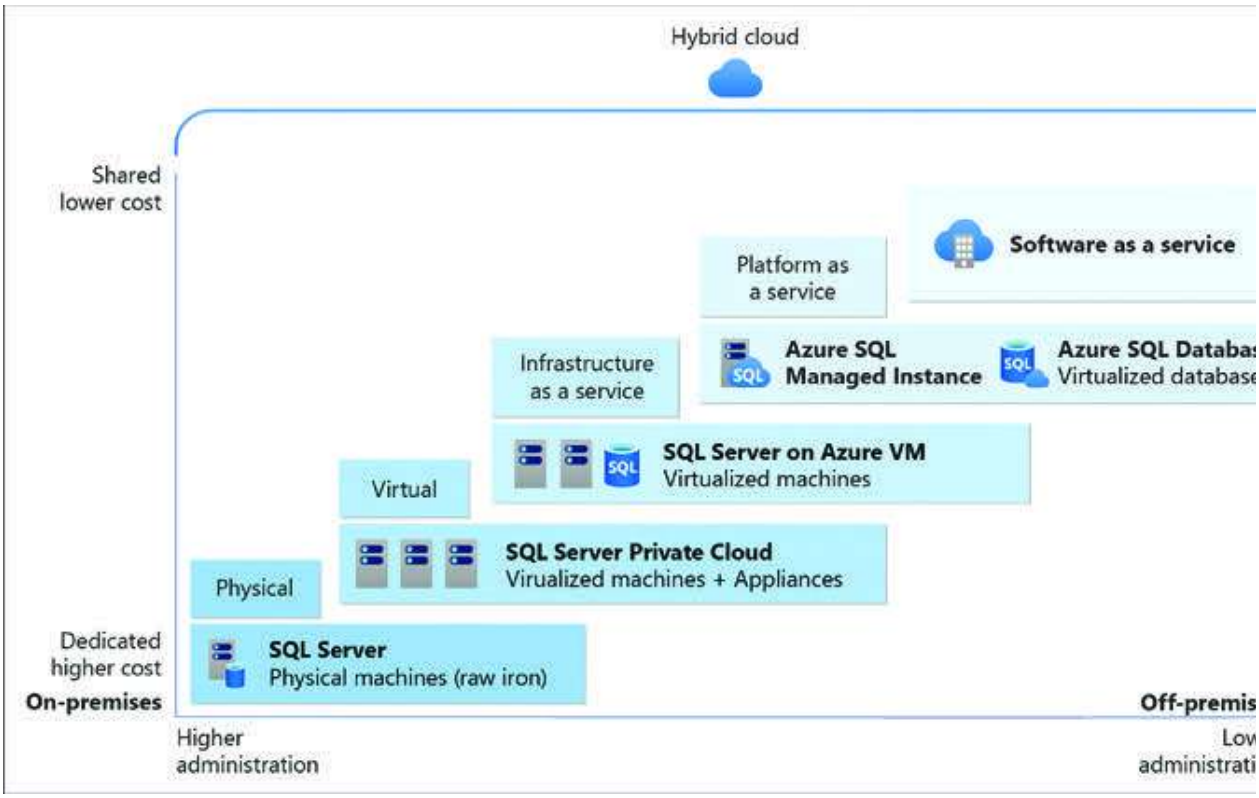


Figure 3.13: Cost versus Administration comparison of various Azure SQL offerings

Now, let’s look at each of the Azure SQL offerings and understand how you can get started with them, starting with SQL Server on Azure Virtual Machines.

SQL Server on Azure Virtual Machines

This is the SQL offering on the cloud you would mostly choose when you are migrating your existing on-premises workloads to the Cloud. It gives you the flexibility to deploy your preferred SQL Server version choice on your preferred hardware, also known as machine sizes and operating system that can be Windows or Linux.

SQL Server on Azure VMs gives you the freedom of almost complete SQL Server administration and removes the hassle of managing and maintaining the required hardware that you would have managed otherwise when deployed in your data centers or on-premises.

You can choose to deploy SQL Server on Azure VMs that are running Windows or Linux Distributions. As you are already aware by now, for Linux you could choose Red Hat Enterprise Linux (RHEL), SUSE Enterprise Linux (SLES) or Ubuntu-based distributions. The licensing modes available for SQL Server Azure VMs are of three types.

The pay-as-you-go model means that the per-second cost of running the Azure VM includes the cost of the SQL Server and OS licenses.

Azure Hybrid Benefit (AHB): This allows you to use your own SQL Server license with a VM that's running SQL Server. So, if you have purchased licenses for SQL Server to deploy on-premises and now you want to use those licenses on Azure, then you will choose the AHB option. This is also called

Bring Your Own Licenses (BYOL) mostly available to Enterprise customers who have the required agreements with Microsoft.

The HA/DR license type is used for the free HA/DR replica in Azure. This again is an option available if you have software assurance agreements with Microsoft, where you can implement a hybrid disaster recovery (DR) plan with SQL Server without incurring additional licensing costs for the passive disaster recovery instance.

For further options on licensing, please refer to the official Microsoft licensing terms available here:

When you are deploying SQL Server Azure VMs, you have the choice to use the PAYG images that have SQL Server pre-installed and ready to be configured, which makes the process easy to get started with SQL Server Azure VMs, as you don't spend time in installing SQL Server.

When creating the SQL Server Azure VMs that use the Windows distribution via the portal you will notice that apart from the general VM settings page, there is an additional SQL Server setting page in the create VM blade as shown in Figure where you can configure the SQL Server based on your requirements, with options like enabling SQL authentication, changing the SQL Server port, Enabling automated patching, Changing the SQL License from payg to AHB and more, as follows.

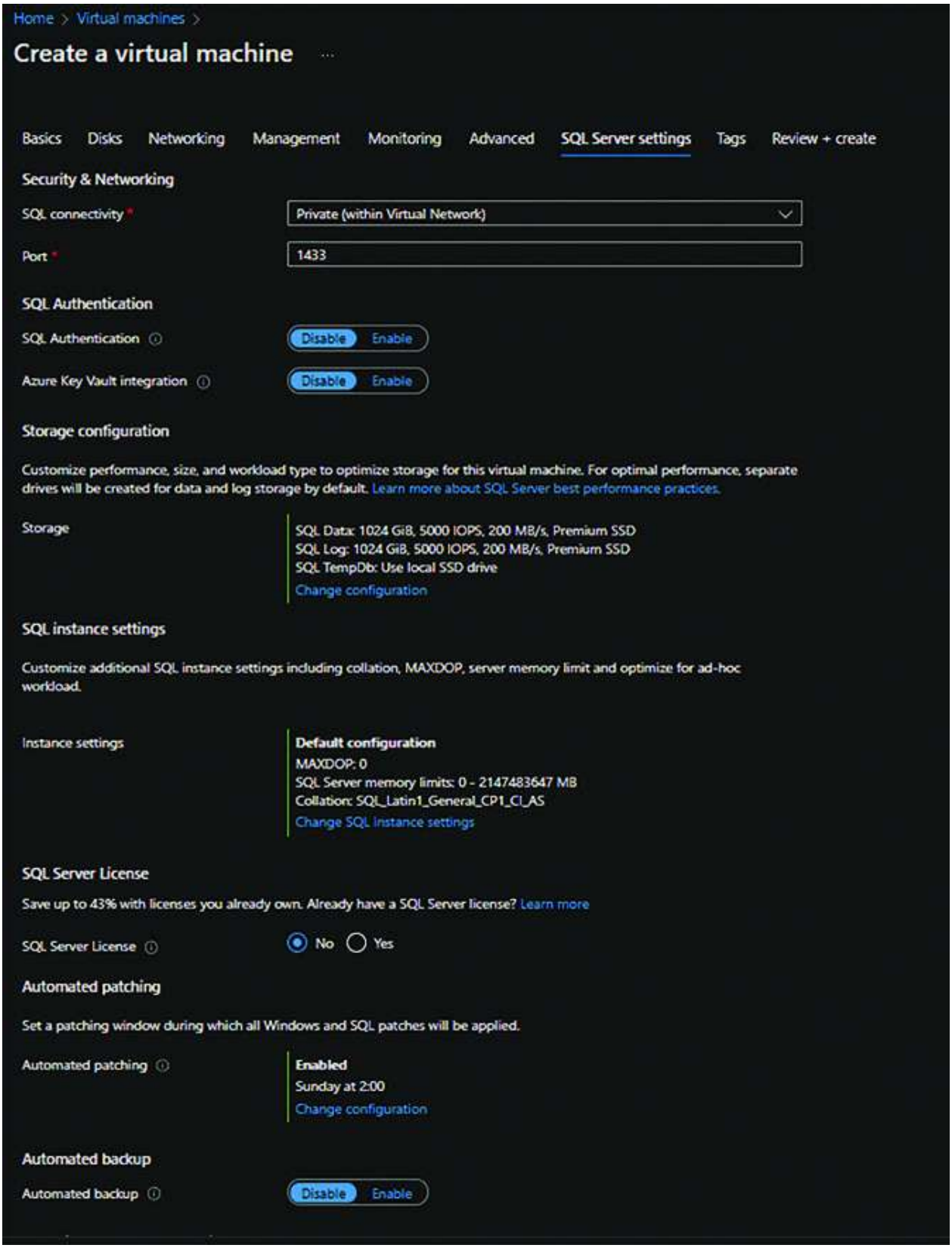


Figure 3.14: SQL Server settings available for SQL Server on Windows Azure virtual machine deployment

This blade of SQL Server settings is not available when you create the SQL Server on Linux-based Azure VMs.

SQL IaaS Agent Extension

Features like Automated patching, Automated backups, Azure Key vault integration, Azure Update Manager integration and changing the license types are made possible using the SQL IaaS agent extension that you can install on your Azure VMs. The SQL IaaS extension is also available for the Azure SQL VMs based on Linux but is currently only available for Ubuntu-based distribution (Not Ubuntu pro).

The SQL Server IaaS agent extension is different from the Azure Arc agents that we spoke about in the earlier sections of this book. The Azure Arc agents are installed on resources outside of Azure, whereas the SQL Server IaaS agent is installed on resources that are created and reside inside Azure.

Also, the IaaS agent for SQL Server on Linux-based Azure VMs is available in lightweight mode, which supports the license management and edition management key features only. You can also use the Azure update manager from the Azure portal to manage your SQL Server on Linux-based Azure VMs. To enable the SQL IaaS extension for a SQL on Ubuntu based Azure VM, you must run the following command:

Enable the Microsoft.SqlVirtualMachine provider for your subscription using a command like:

```
# Register the SQL IaaS Agent extension to your subscription
az provider register --namespace Microsoft.SqlVirtualMachine
```

Register the VM with the SQL IaaS extension agent using the command such as:

```
# Register Enterprise or Standard self-installed VM in Lightweight mode
az sql vm create --name --resource-group --location --license-type
```

You can verify that the IaaS extension agent is successfully installed by checking in the portal, in the Azure portal you can see your VM enlisted under the SQL virtual machine resources, as shown in Figure You can see the option to configure the licensing for the SQL Server machines, this shows the extension was successfully registered.

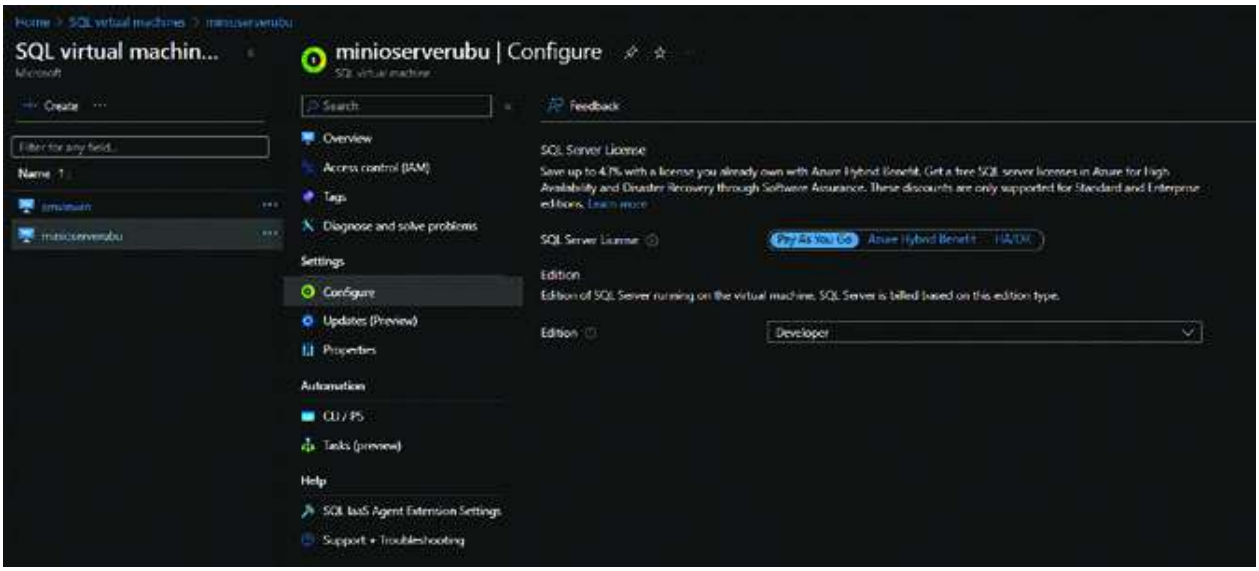


Figure 3.15: IaaS extension for SQL Server on Linux-based Azure VM deployments

If you are creating the Azure SQL VMs based on Windows from the Azure Marketplace image through the portal, then during the creation of the VM itself, the extension is registered. If you choose to do a self-install of SQL

Server on the Azure Virtual Machine, then you must register your SQL Server CM with the SQL IaaS agent extension manually. Also, if you have the CEIP service running for Azure SQL on Windows VM for SQL Server 2016 or later, then the VM is automatically registered with the SQL IaaS agent extension. All of this is true only for the SQL Server on Windows-based Azure VMs.

Figure 3.16 shows one of the SQL servers on Windows-based Azure VM that has the extension registered and, in the portal. We can see that VM under the SQL virtual machine resources. Notice that for SQL Server on Windows-based Azure VM, the SQL IaaS extension provides more options like storage configuration, Backups, SQL best practice assessment and more.

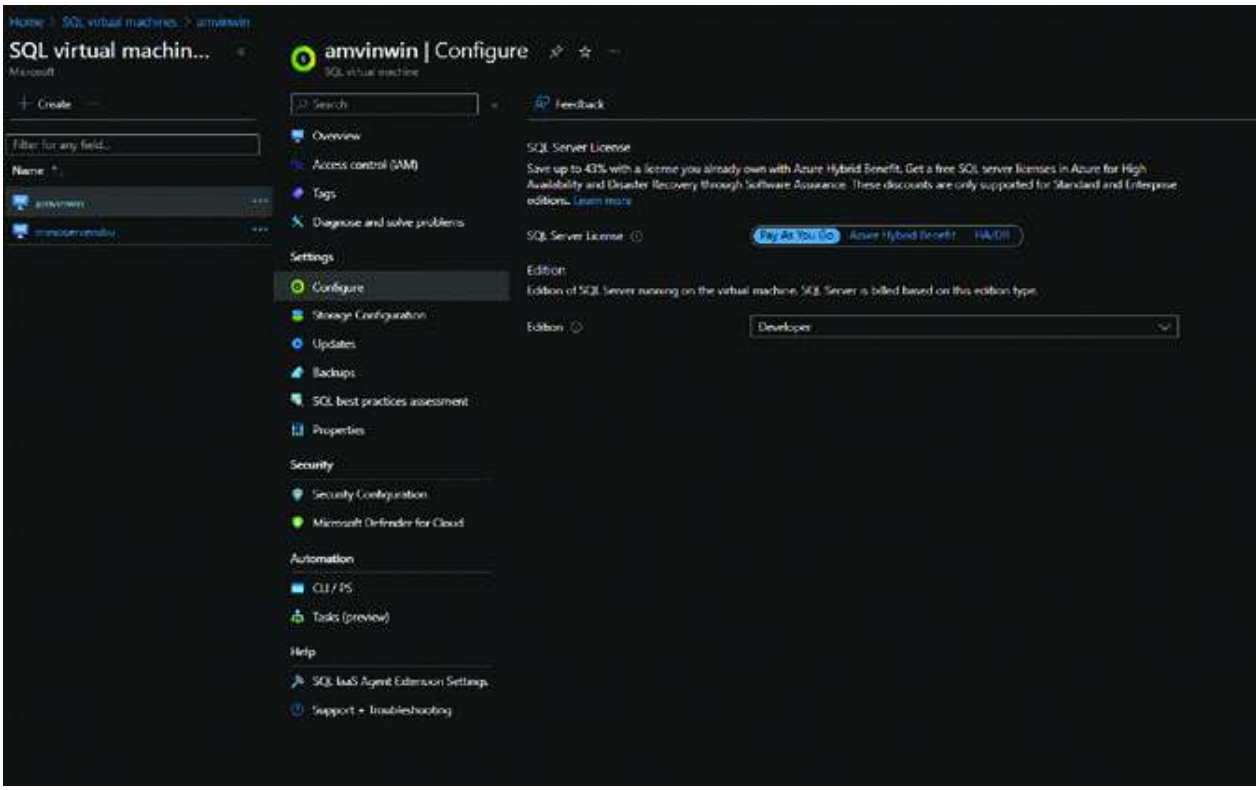


Figure 3.16: IaaS extension for SQL Server on Windows-based Azure VM deployments

The steps to manually register the SQL IaaS Extension agent are the same as for Linux. You first register the resource provider Microsoft.SqlVirtualMachine in case it is not registered for the subscription using the command:

```
# Register the SQL IaaS Agent extension to your subscription
Register-AzResourceProvider -ProviderNamespace
Microsoft.SqlVirtualMachine
```

Then manually register the extension using the command:

```
# Get the existing Compute VM

$vm = Get-AzVM -Name -ResourceGroupName

New-AzSqlVM -Name $vm.Name -ResourceGroupName
$vm.ResourceGroupName -Location $vm.Location -LicenseType
```

In the next chapters, we will delve into detail the best practices to be followed when deploying SQL Server VMs and setting up business continuity.

Azure SQL Database (SQLDB)

Azure SQL Database is the fully managed platform as a service (PaaS) database service provided by Microsoft. It runs an evergreen version of SQL Server, meaning that it runs the latest stable version of the SQL Server database engine and patched OS and provides 99.99% availability.

If you are developing a new cloud-based modern application that requires relational data and non-relational data structure support such as JSON, XML, and graph, then you can choose Azure SQL DB. It has the engine that powers the data layer for your application.

When you are creating an Azure SQL DB, it is not an instance of SQL Server that gets created, but you get a database to which you can connect. There are two purchase models available for Azure SQL DB and they are:

vCore-based purchasing model, which lets you choose the number of vcores (virtual cores), amount of memory and storage, including the speed of the storage. Using this model, you can enable the Azure Hybrid Benefit for SQL Server to use your existing SQL Server licensing as you did with the Azure SQL VMs.

There are three tiers in vCore-based purchasing, and they are:

General Budget-oriented option that provides balanced computing and storage options.

Business Critical: Designed for OLTP applications with high transaction rates and low I/O latency requirements.

Hyperscale: This model provides independently scalable compute and storage resources. Generally recommended for most of the business workloads that intend to run production workloads.

Another simple licensing option is the Database Transaction Unit (DTU) based purchasing model that offers a mix of compute, memory and I/O resources. This licensing model is also tier-based and has only two tiers, including:

Standard: Designed for most common workloads. Budget-oriented balanced compute and storage options.

Mostly used for OLTP applications with high transaction rates that require low I/O latency.

If you choose the vCore-based purchasing model, you also get the option where you can choose the compute tiers. There are two compute tiers available and they are:

Provisioned compute Provides a specific amount of resources that is continuously provisioned independent of workload activity, and bills for the amount of compute provisioned at a fixed price.

The Serverless compute tier automatically scales compute resources based on workload activity and bills for compute used per second.

Here is a demo for you to get started with the Azure SQL DB. The first step is to log into the Azure portal and then search for SQL DB, which should take you to the create SQL database blade, as shown in Figure

Home > SQL databases > Create SQL Database >

Create SQL Database ...

Microsoft

Create a SQL database with your preferred configurations. Complete the Basics tab then go to Review + Create to provision with smart defaults, or visit each tab to customize. Learn more

Did you know that new users in Azure can create a free Azure SQL Database and use it for 12 months using Azure free account? Learn more

Project details

Select the subscription to manage deployed resources and costs. Use resource groups like folders to organize and manage all your resources.

Subscription * : Visual Studio Enterprise Subscription

Resource group * : (New) sqldatabases

Create new

Database details

Enter required settings for this database, including picking a logical server and configuring the compute and storage resources

Database name * : sqldbtest

Server * : (new) amvintestserver (Central India)

Create new

Want to use SQL elastic pool? : ○ Yes ◉ No

Workload environment : ◉ Development ○ Production

Default settings provided for Development workloads. Configurations can be modified as needed.

Compute + storage * : **Basic**
2 GB storage
Configure database

Backup storage redundancy

Choose how your PITR and LTR backups are replicated. Geo restore or ability to recover from regional outage is only available when geo-redundant storage is selected.

Backup storage redundancy : ◉ Locally-redundant backup storage ○ Zone-redundant backup storage ○ Geo-redundant backup storage

Review + create | Next : Networking >

Figure 3.17: Azure SQL Database creation from Azure Portal

If you are creating the Azure SQL DB first time, then you must create a logical server for the first time, click the “create new” option for the server option shown here. This will open a blade as shown in Figure provide the server’s name, location and other details based on your requirement.

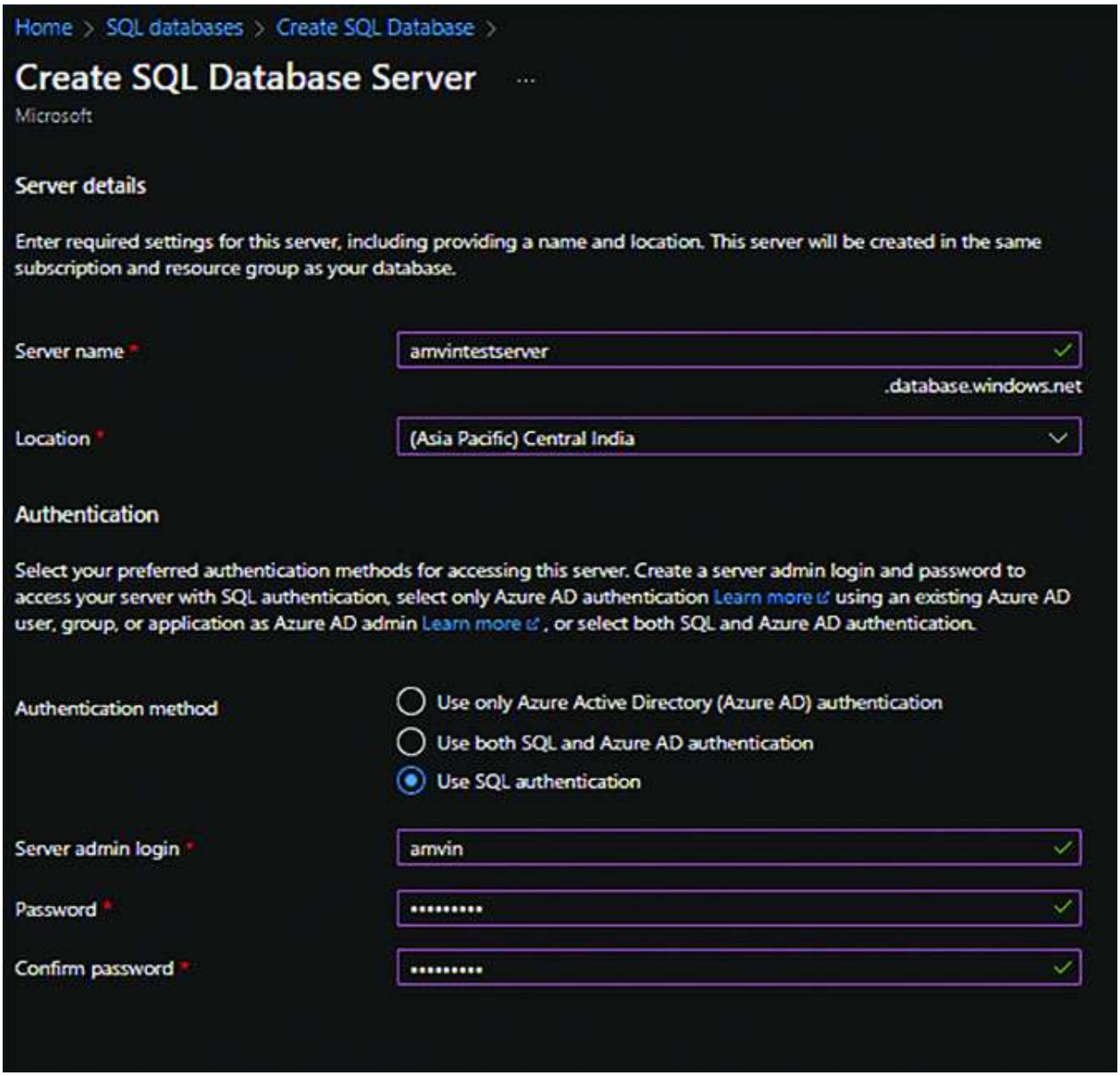


Figure 3.18: Azure SQL database details provided in the Azure portal

Now to select the licensing model between DTU versus vCore, in the create SQL database blade, click the configure database option under the compute+storage selection and in the drop-down. You will see two sections, DTU or vCore based and then the tiers as shown in Figure

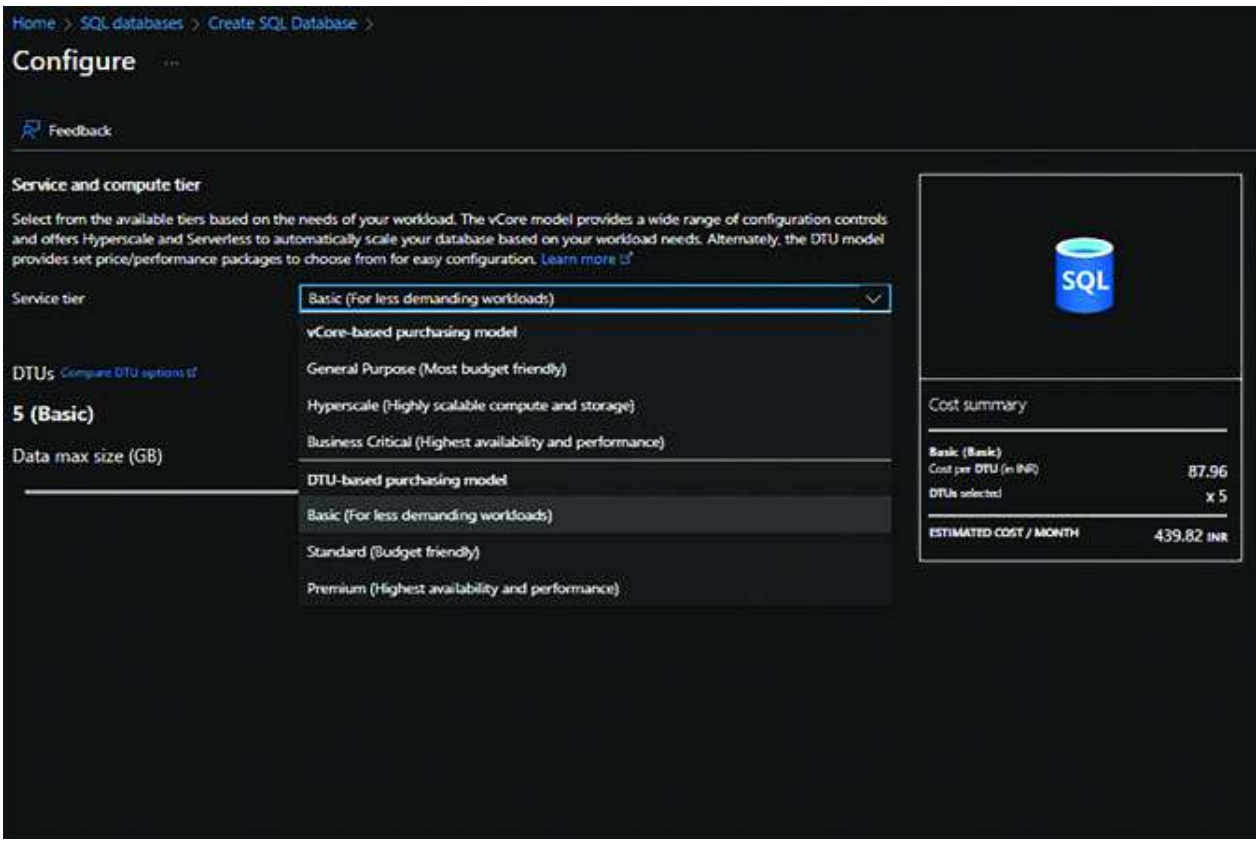


Figure 3.19: Azure SQL database licensing model options

When you choose the vCore options, you can configure the number of vCores, database max size and the compute tiers as shown in Figure

Figure 3.20: Azure SQL database vCore option-based compute tiers

When you choose the DTU model, none of these options are available as shown in Figure

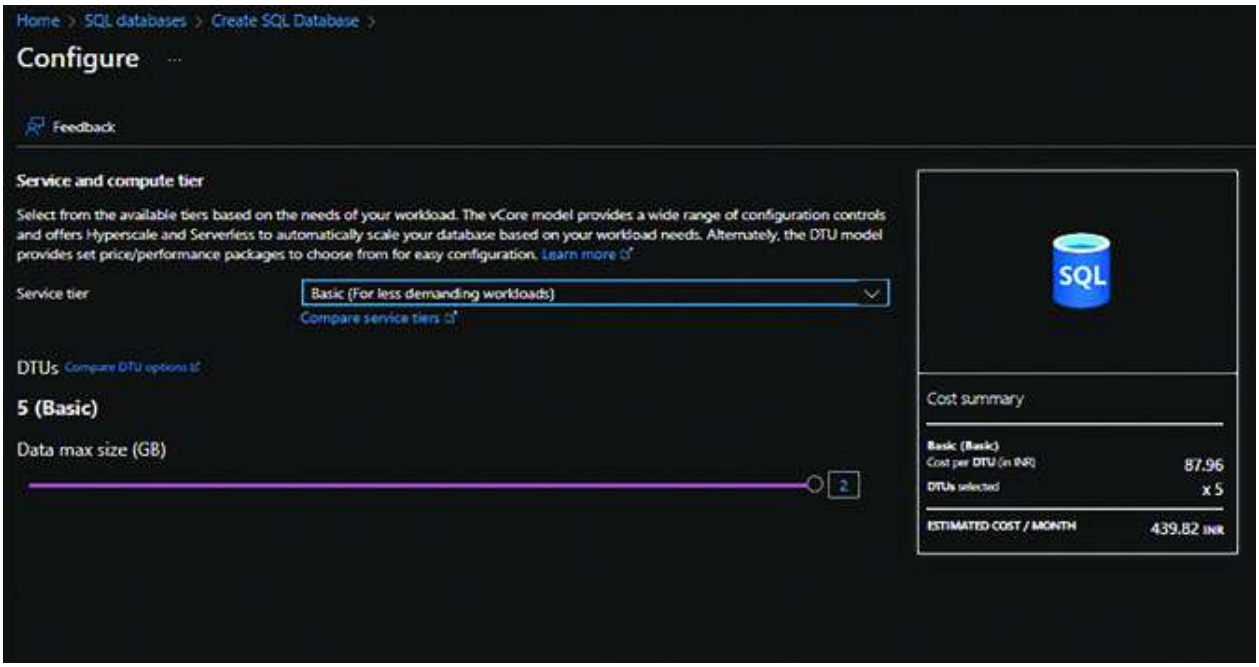


Figure 3.21: Azure SQL database-based DTU model configuration options

You can continue following the instructions for the creation of the SQL DB in the portal including the database collations, connectivity information, ledger configuration, always encrypted option and more, when the SQL DB is created it shows up in the Azure portal under the SQL database blade, as shown in Figure

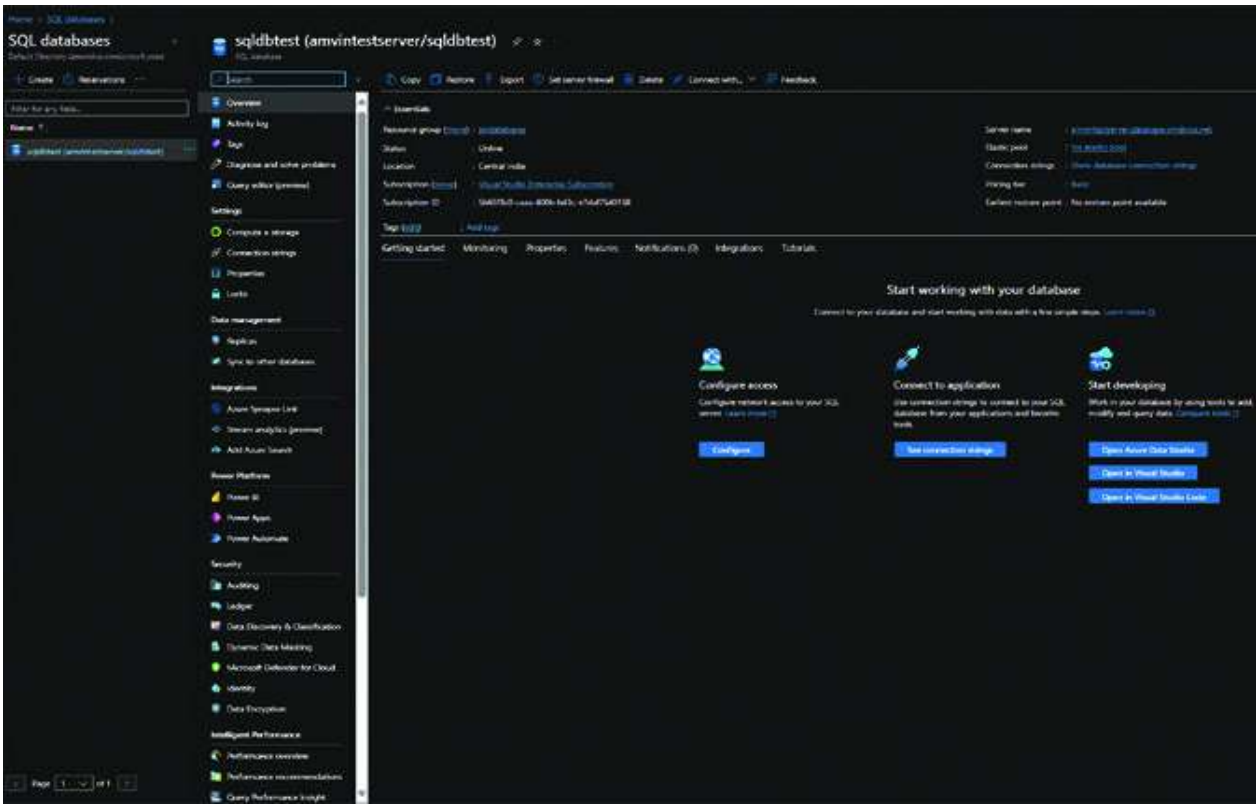


Figure 3.22: Azure SQL Database as shown in the Azure portal

You can connect to the Azure SQL DB using tools like SSMS and ADS. Figure 3.23 is a screenshot for the same. Before connecting, you need to ensure that you set the logical server for public access and add the client IP in the allow list:

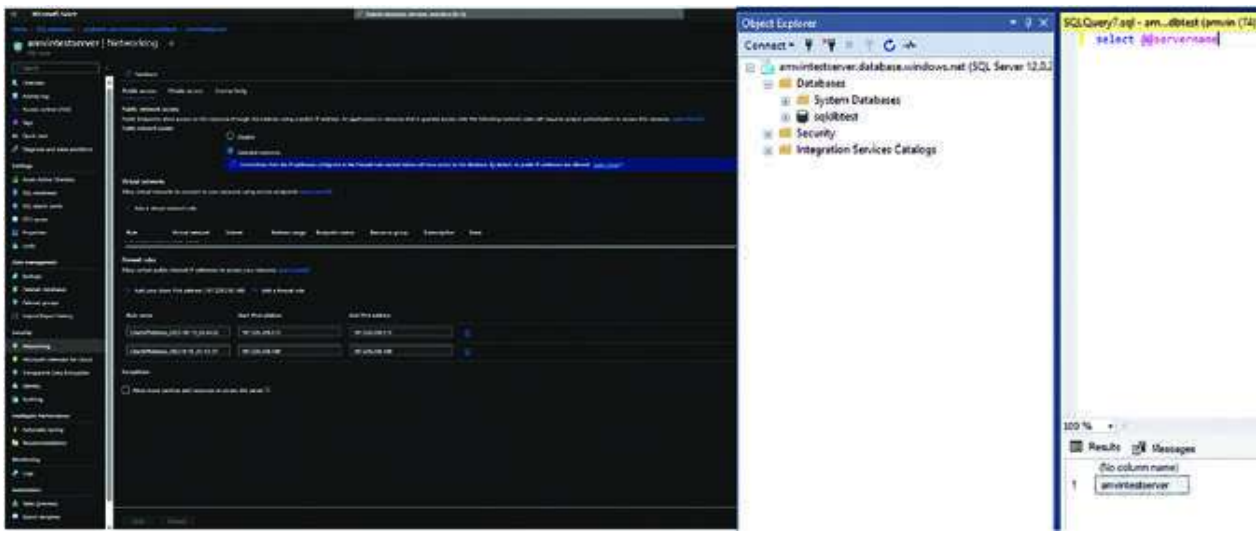


Figure 3.23: Connecting to Azure SQL DB using the SSMS

We will talk about best practices for implementation and features like ledger in the upcoming chapters.

Azure SQL Managed Instance

Azure SQL DB provides a database as a platform as a service (PaaS), but if you are looking to get an entire SQL Server instance as a service, then Azure SQL Managed Instance is the cloud offering that you should be going with. It is a fully managed platform as a Service with the same 99.99% availability as Azure SQL DB. Also, it is almost 100% compatible with the latest Enterprise edition of the SQL Server. If you are looking to migrate a larger number of applications from on-premises or IaaS to a fully managed PaaS cloud environment, with low migration effort then this is the platform of your choice, as you can simply perform a lift and shift migration of the databases to cloud.

Azure SQL Managed Instance comes with a native virtual network implementation to address the common security concerns for existing SQL Server customers. This provides a native virtual network implementation and connectivity to on-premises environments over the Azure ExpressRoute or VPN gateway, providing the required security isolation needed for environments.

vCore-based purchasing model with Azure SQL Managed Instance is the only available purchase model. Under the vCore purchase model, you can choose various hardware configurations. This includes the Standard series, Premium series and the Premium Series memory-optimized, as shown in Figure

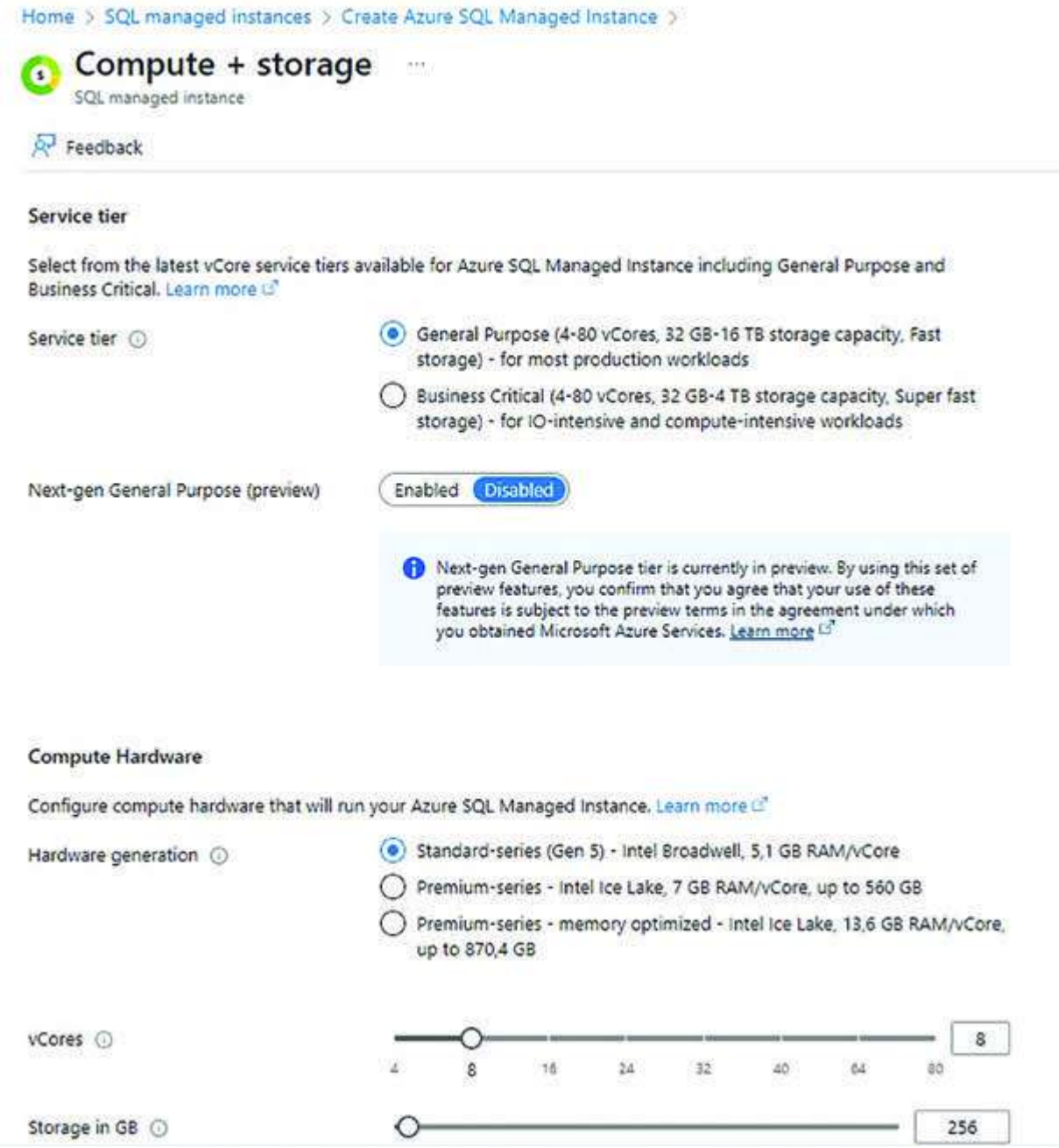


Figure 3.24: Different Service tiers and compute hardware options for Azure SQL MI

You can also choose the service tiers as shown here between General Purpose - typically used for normal performance I/O latency requirements) or the Business-Critical tier – used for workloads that require low I/O latencies and highest performance.

You can create the Azure SQL Managed Instance as shown from the portal blade:

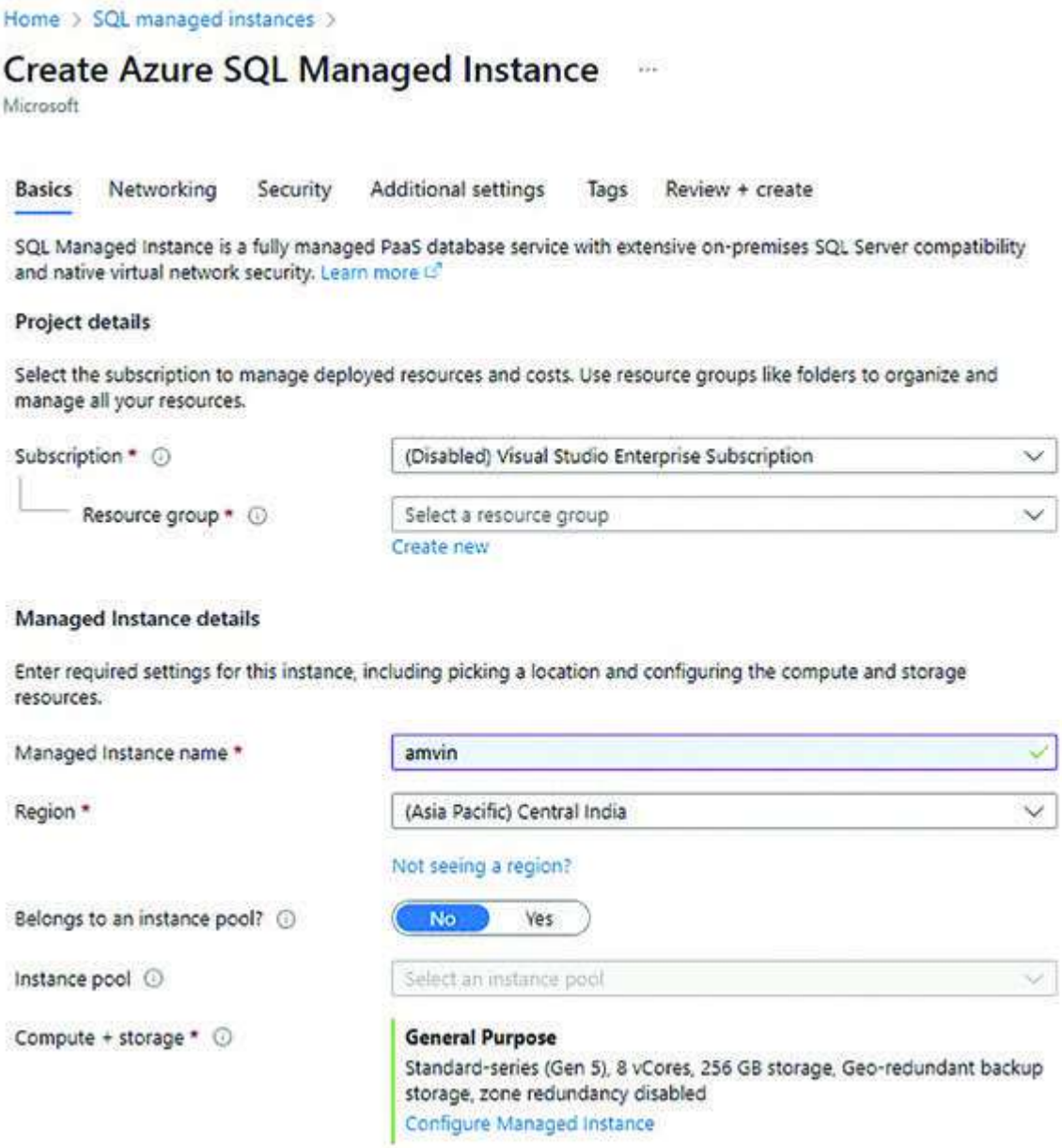


Figure 3.25: Various options to provide while creating the Azure SQL Managed Instance in Azure Portal

It provides you with the option of setting up the collation, authentication method, encryption options and others like Azure SQL DB. Once the instance is created, if you intend to connect to it from an on-premises environment, then you require to connect your on-premises application to the Vnet-local endpoint of your SQL Managed Instance. That requires a site-to-site connection between the application and the SQL Managed Instance virtual network.

Azure SQL Edge

Azure SQL Edge is the optimized relation database engine that is made available for IoT and IoT edge applications. It provides stream, process and analyzes relation and non-relation data like JSON, XML, and time-series data, thus making it the right choice for a variety of modern IoT applications.

It supports two deployment modes, including:

Connected deployment through Azure IoT edge: In this scenario, Azure SQL edge is deployed from the Azure Marketplace and deployed as a module for Azure IoT edge.

Disconnected Deployment: Azure SQL Edge container images can be pulled from the docker hub and deployed as a container or on a Kubernetes cluster.

For Azure SQL Edge, you only have two editions available and they are:

Azure SQL edge developer: Development only sku limited to 4 cores and 32 GB memory.

Azure SQL edge: Production sku, each SQL edge container is limited to 8 cores and 64 GB of memory.

In this book, we will not focus on Azure SQL edge as it focuses on full-fledged SQL Server deployments used for non-IoT scenarios specifically for production-grade workloads running OLTP, OLAP and HTAP workloads.

If you want to know more about Azure SQL Edge and how you can get started. please refer to the official Microsoft documentation:

Conclusion

In this chapter, we learned how we can get started with SQL Server on Linux, and also discussed the SQL Server on Linux general architecture that included the Linux SQL Server process detail as well.

We then went through a hands-on demo lab with step-by-step information about how you can obtain and install the SQL Server packages.

Later, we also focused our attention on getting started with SQL Server containers on both standalone and Kubernetes environments, here also we looked at a deployment of SQL Server container demo.

Finally, we discussed the Azure SQL family, which included a discussion on the SQL Server on the Azure Virtual Machine, SQL IaaS extension and benefits of the same, Azure SQL database, Azure SQL Managed Instance and Azure SQL edge. For most of these topics, we covered the deployment and licensing options, which hopefully helps you in selecting the right choice of SQL offering that suits your requirement the best.

In the next chapter, we will explore various aspects of SQL Server security, covering topics such as authentication and authorization, encryption, key management, securing data transmission, threat detection and response, and so on.

Additional Resources

Getting Started with SELinux

SQL Server on Linux- Introduction

SQL Server PAL -

Microsoft Packages

Microsoft Artifact Registry:

SQL Server Docker hub

MobaXterm

Deploy SQL Server – The Ansible way! - Microsoft Community Hub

Microsoft SQL Server official repository - SLES example (github.com)

Microsoft SQL Server official repository - Preview Repo (github.com)

What is Azure SQL? - Azure SQL | Microsoft Learn

Microsoft SQL Server Licensing

SQL-Server-Unveiled---Path-to-Data-Excellence/Chapter 3 at main · ava-orange-education/SQL-Server-Unveiled---Path-to-Data-Excellence (github.com)

Microsoft SQL Edge

CHAPTER 4

SQL Server and Security Hand in Glove

Introduction

In today's digital landscape, data security is paramount. With the increasing sophistication of cyber threats, it's crucial for organizations to implement robust security measures to safeguard their sensitive information. SQL Server, as a leading relational database management system, offers a range of features and tools to enhance security and protect valuable data assets. In this chapter, we will delve into various aspects of SQL Server security, covering topics such as authentication and authorization, encryption, key management, securing data transmission, threat detection and response, row-level security, and best practices.

By mastering the concepts and techniques discussed in this chapter, readers will achieve a comprehensive understanding of how to fortify their SQL Server environments against potential security breaches. You will learn how to authenticate and authorize users effectively, implement encryption to safeguard data at rest and in transit, manage encryption keys securely, enhance data protection during transmission, detect and respond to security threats efficiently, enforce row-level security to control access to specific rows in a database, and adhere to best practices to ensure optimal security posture.

Structure

In this chapter, we will cover the following topics:

Evolution of Security Features and Current Trends in SQL Server

Authentication and Authorization

SQL Server Encryption

Key Management in SQL Server

Connection Security

Threat Detection and Response

Best Practices to Securing SQL Server and Azure SQL Database

Evolution of Security Features in SQL Server

Security in SQL Server has undergone a remarkable evolution over the years, adapting to the ever-changing landscape of data threats and vulnerabilities. As organizations increasingly rely on databases to store and manage sensitive information, SQL Server has continually enhanced its security arsenal to provide robust protection for data assets. This journey from rudimentary security measures to sophisticated, multi-layered defenses reflects the commitment of Microsoft to address the evolving challenges in the realm of data security.

Early Years: SQL Server 2000 to 2008

In its nascent stages, SQL Server focused on foundational security measures, primarily centered around authentication and basic access control.

SQL Server 2000 introduced the distinction between Windows Authentication and SQL Server Authentication, offering flexibility in user identity verification. However, the emphasis was on securing the database engine itself, with limited provisions for comprehensive security policies.

With the release of SQL Server 2005, a pivotal shift occurred with the introduction of the SQL Server Management Studio (SSMS) and an enhanced security model. Role-based security became more robust, enabling administrators to define and manage permissions more granularly. This laid the groundwork for a more mature security infrastructure.

SQL Server 2008 continued this trajectory, refining encryption capabilities and introducing auditing features. Transparent Data Encryption (TDE) made its debut, offering a straightforward method to encrypt entire databases, bolstering data-at-rest security.

Advancements in the 2010s: SQL Server 2012 to 2022

As cyber threats became more sophisticated, SQL Server responded with heightened security measures. SQL Server 2012 marked the introduction of AlwaysOn Availability Groups, providing both high availability and enhanced security through automatic failover and readable secondary replicas.

SQL Server 2016 introduced Always Encrypted, a groundbreaking feature that allowed sensitive data to remain encrypted even during processing, mitigating risks associated with insider threats. Dynamic Data Masking was also added to limit sensitive data exposure in real-time based on user roles.

The 2017 release brought advancements in SQL Server on Linux, extending security features to a broader range of platforms. Moreover, it continued to refine existing features, ensuring a consistent security posture across diverse environments.

In SQL Server 2019, Microsoft embraced the principles of Confidential Computing with the introduction of Secure Enclaves. This innovative technology allowed sensitive data to be processed in an isolated, secure area within the database, safeguarding against even privileged access threats.

SQL Server 2022 introduces key security enhancements, including support for Azure Active Directory Authentication for streamlined user access, Always Encrypted with Secure Enclaves enabling secure and complex queries, and the ability to backup and restore certificates using the PFX format for simplified certificate management. The default cryptography mechanisms have been improved to address evolving threats, while hypervisor security and measures against firmware attacks provide additional layers of protection. Enhanced Server Message Block (SMB) features, supporting SMB over QUIC and East-West Encryption, contribute to more efficient and secure data transfer.

Collectively, these updates bolster SQL Server's security posture, reflecting Microsoft's commitment to providing a robust and secure database platform against contemporary threats.

Current Trends: Cloud Integration and Beyond

The current era sees SQL Server tightly integrated with cloud platforms, enabling organizations to leverage advanced security services. Azure Active Directory authentication and cloud-native security features have become integral components of SQL Server's security ecosystem. Additionally, the adoption of Threat Detection and Advanced Threat Protection further demonstrates Microsoft's commitment to proactive security measures.

In conclusion, the evolution of security features in SQL Server mirrors the broader evolution of data security in the digital age. From foundational measures to sophisticated encryption and cloud-native security, SQL Server continues to adapt, providing organizations with a robust defense against the ever-evolving landscape of cyber threats. As the SQL Server ecosystem evolves, so too will its commitment to ensuring that data remains secure, making it a key player in the ongoing quest for data protection.

The Who and the What – Authentication and Authorization

Navigating the realm of database security begins with understanding the roles of authentication and authorization. Authentication mechanisms verify the identities of users seeking access to a SQL Server environment. Once authenticated, the authorization process determines the level of access and privileges granted to individuals or entities within the database system.

Authentication in SQL Server

Authentication is the process of verifying the identity of users or applications attempting to access a SQL Server instance. It ensures that only authorized entities can connect to the database. SQL Server supports multiple authentication methods, each with its own characteristics.

Windows Authentication relies on the Windows operating system for user authentication. Users log in using their Windows credentials (username and password). It provides a seamless experience for users who are part of the same Windows domain as the SQL Server. The major advantage of using Windows Authentication is the strong security it provides that leverages Windows security features. It also allows single sign-on (SSO) for which users don’t need to enter separate credentials. There are some considerations of using Windows authentication that are important to remember, including the requirement of a Windows domain environment and it is not available for non-Windows clients.

Figure 4.1: Windows Authentication

SQL Server Authentication (Mixed SQL Server Authentication allows users to log in using SQL Server-specific credentials (username and password). The credentials are stored within SQL Server itself, in the master database of the SQL Server instance. The major advantages of using SQL Server authentication are that it works across different platforms (Windows, Linux, and more) and is useful for applications that don't support Windows Authentication. The most important considerations a DBA must be aware of using SQL Server authentication are that the passwords are stored in the SQL Server instance (which makes it less secure) and it requires managing separate credentials.

Figure 4.2: SQL Authentication

Microsoft Entra Microsoft Entra ID (formerly also known as Azure Active Directory) is Azure's cloud-based identity and access management service. Microsoft Entra ID is similar to Active Directory, which provides a centralized repository for managing access to organization's resources. Identities are objects in Microsoft Entra ID that represent users, groups, or applications. They can be assigned permissions through role-based access control and be used for authentication to Azure resources. Microsoft Entra authentication is supported for:

Azure SQL Database

Azure SQL Managed Instance

SQL Server 2022 on Windows Azure VMs

Azure Synapse Analytics

SQL Server 2022 with a supported Windows or Linux operating system

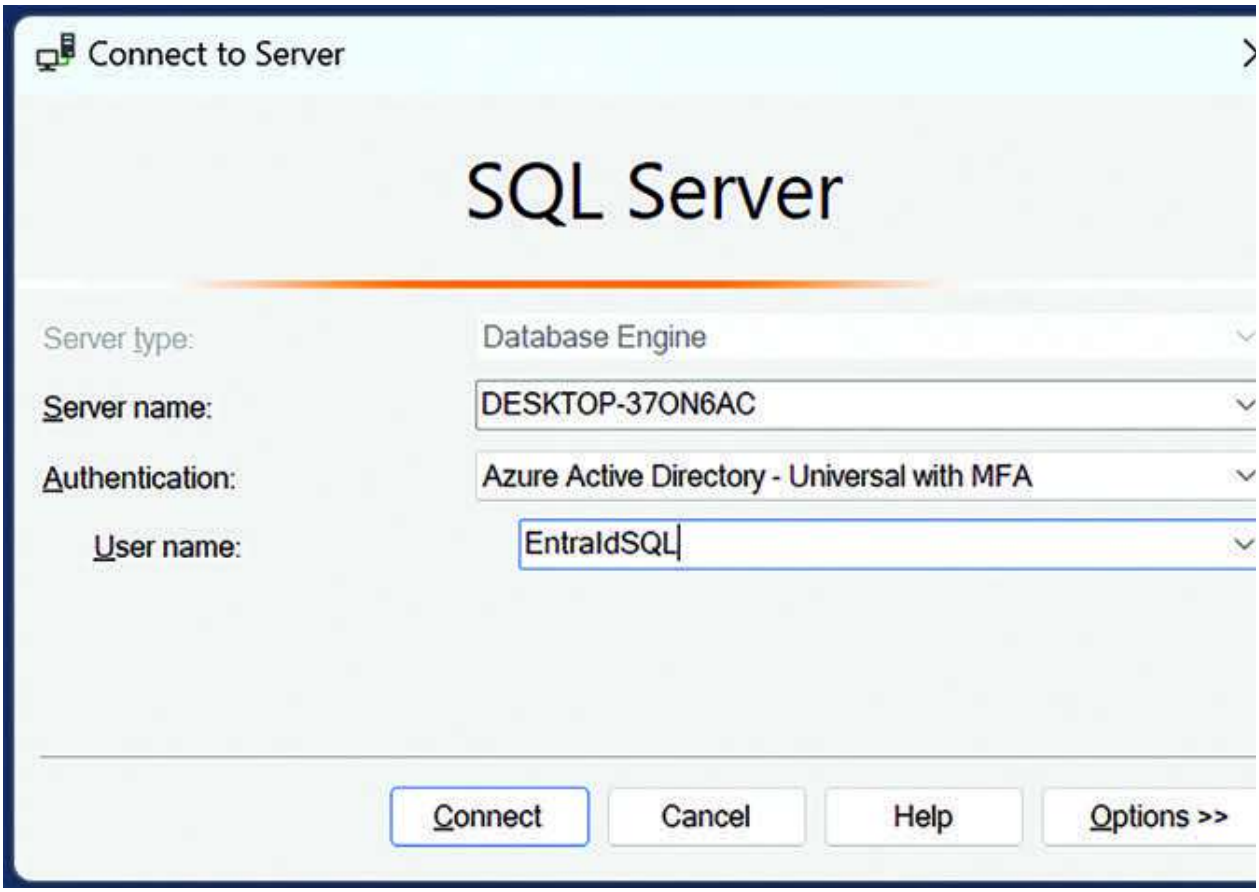


Figure 4.3: Microsoft Entra Authentication

When your Windows Server Active Directory is federated with Microsoft Entra ID, users can authenticate with SQL Server using their Windows credentials. This authentication can occur either as Windows logins or Microsoft Entra logins. However, it's important to note that Microsoft Entra ID does not fully support all the features available in Windows Server Active

Directory. For instance, features like service accounts or complex networking forest architecture are not fully compatible with Microsoft Entra ID. On the other hand, Microsoft Entra ID offers additional capabilities, such as multifactor which are not available in the traditional Active Directory.

By leveraging Microsoft Entra authentication, you gain the ability to centrally manage the identities of database users and other Microsoft services. This centralized identity management provides a convenient way to handle database user management and simplifies permission management across various services. Benefits include the following:

Microsoft Entra provides an alternative to traditional SQL Server authentication methods.

It helps prevent the proliferation of user identities across multiple servers.

Microsoft Entra allows for centralized password rotation, enhancing security.

Customers can manage database permissions using Microsoft Entra groups.

By enabling integrated Windows authentication and other supported methods, Microsoft Entra eliminates the need to store passwords.

Microsoft Entra authentication uses contained database users for identity verification at the database level.

Microsoft Entra ID supports token-based authentication for applications connecting to SQL Database and SQL Managed Instance.

SQL Server Management Studio supports connections using Microsoft Entra with multi-factor authentication, enhancing security by requiring multiple forms of verification during login. Users have a range of convenient choices, such as phone calls, text messages, smart cards with PINs, or mobile app notifications. This layered approach significantly strengthens the authentication process, safeguarding sensitive accounts and data.

There are various scenarios that an organization may fall into and requires certain considerations, including:

Only the cloud-based components of Microsoft Entra ID, including SQL Database, SQL Managed Instance, SQL Server on Windows Azure VMs, and Azure Synapse are considered to support Microsoft Entra native user passwords.

To enable Windows single sign-on credentials (or user/password for Windows credential), utilize Microsoft Entra credentials from a federated or managed domain configured for flawless single sign-on for pass-through and password hash authentication.

Communication with the ADFS block is necessary to support Federated authentication (or user/password for Windows credentials).

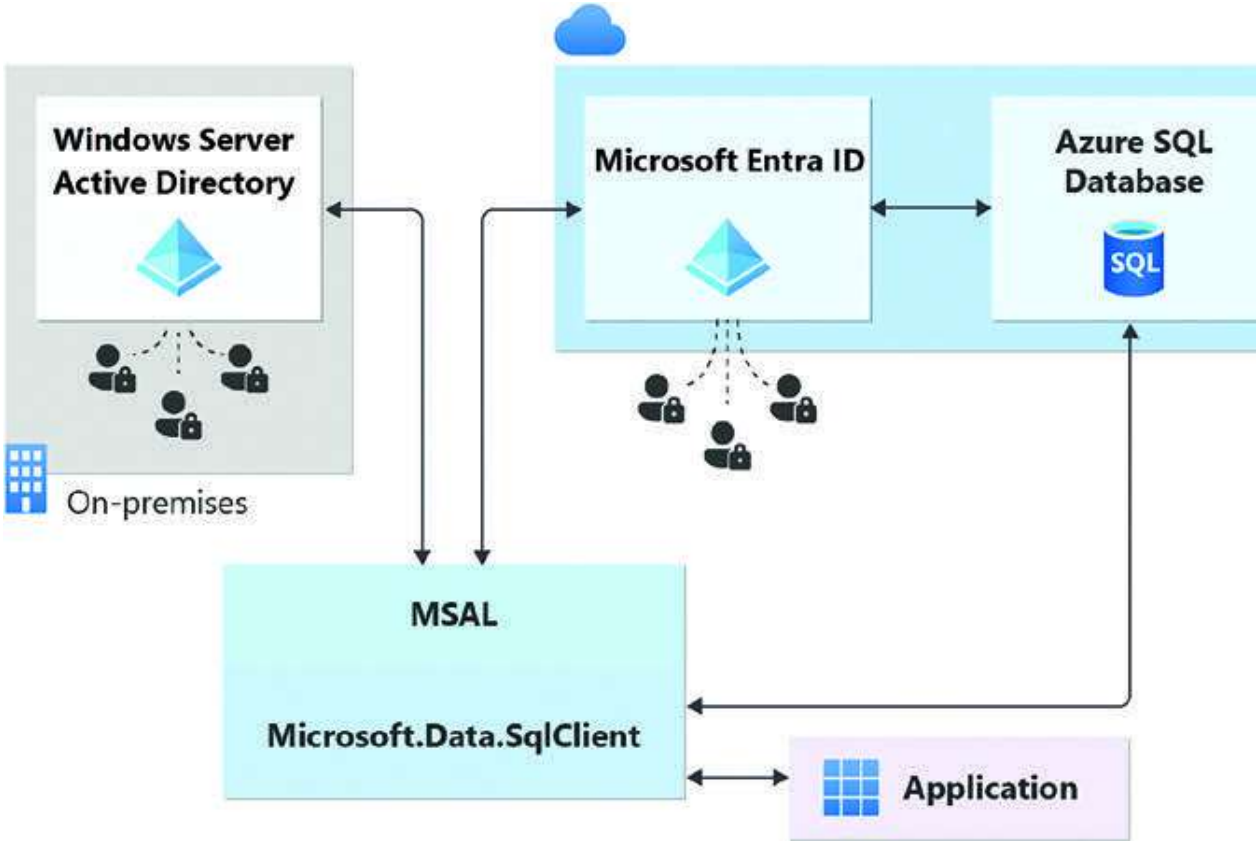


Figure 4.4: Sample federated authentication with ADFS infrastructure (or user/password for Windows credentials)

For Azure SQL, Azure VMs, and SQL Server 2022, Microsoft Entra authentication only supports access tokens which originate from Microsoft Entra ID and doesn't support third-party access tokens. Microsoft Entra ID also doesn't support redirecting Microsoft Entra ID queries to third-party endpoints. This applies to all SQL platforms and all operating systems that support Microsoft Entra authentication.

Authorization and Access Control

In SQL Server 2022 and Azure SQL Database, authorization and access control have evolved to meet the increasing demands of secure data management. The integration with Azure Active Directory, advancements in role-based security, fine-grained access control through permissions, and the introduction of Dynamic Data Masking collectively contribute to a comprehensive and adaptive security framework. These features empower organizations to tailor access control policies to their specific requirements, ensuring a secure and compliant database environment in both on-premises and cloud-based scenarios.

Role-Based Security:

In SQL Server 2022 and Azure SQL Database, role-based security remains a foundational element of access control. Roles enable the logical grouping of users and permissions, simplifying the management of access privileges. With advancements in SQL Server 2022, role-based security has become more robust, allowing administrators to create custom roles tailored to specific organizational needs. Azure SQL Database, being a cloud-based service, seamlessly integrates with Azure Active Directory, providing enhanced role-based access control through the Azure AD authentication model. This integration facilitates efficient user management and access control across the Azure ecosystem.

Consider an e-commerce platform with various user roles such as “Customer,” “Sales Representative,” and “Administrator.” Each role should have specific access permissions based on their responsibilities. In SQL Server 2022 or Azure SQL Database, an administrator creates roles and assigns permissions. The “Sales Representative” role might have permissions to view and update customer orders, while the “Administrator” role has broader access to manage products, customer data, and system configurations. Users are then assigned to these roles, ensuring that each user has an appropriate level of access aligned with their role within the organization.

```
-- Role Creation
CREATE ROLE SalesRepresentative;

CREATE ROLE Administrator;

-- Assign Permissions to Roles
GRANT SELECT, UPDATE ON Orders TO SalesRepresentative;
GRANT SELECT, UPDATE, INSERT, DELETE ON Products, Customers
TO Administrator;

-- User Assignment to Roles
EXEC sp_addrolemember 'SalesRepresentative', 'JohnDoe';
EXEC sp_addrolemember 'Administrator', 'JaneAdmin';
```

Fine-Grained Access Control through Permissions:

SQL Server 2022 continues to offer fine-grained access control mechanisms through permissions. Database administrators can precisely define what actions users or roles are allowed to perform at both the database and object levels. In Azure SQL Database, this granular control extends to the server and database resources, ensuring that access is strictly aligned with organizational security policies. Additionally, advancements in SQL Server 2022 introduce more sophisticated permission sets, allowing for nuanced control over specific data operations. This fine-grained access control is crucial for organizations aiming to maintain a secure and compliant database environment.

In a healthcare database, it is essential to control access to sensitive patient records. Different users may need varying levels of access based on their roles, ensuring compliance with privacy regulations. Using SQL Server 2022

or Azure SQL Database, administrators can grant specific permissions to users on sensitive tables. For instance, only medical staff with the role might have permission to view patient diagnosis details, while administrative staff with the role “Clerk” may only have access to non-sensitive patient information.

```
-- Grant Permissions on Patient Records
GRANT SELECT (NonSensitiveData) ON Patients TO Clerk;
GRANT SELECT (DiagnosisDetails) ON Patients TO Doctor;
```

Introduction of Dynamic Data Masking for Enhanced Authorization:

Dynamic Data Masking (DDM) is a notable feature introduced for enhanced authorization in SQL Server 2022 and Azure SQL Database. DDM allows administrators to define masking rules on sensitive data, ensuring that unauthorized users see only a masked or obfuscated version of the information. This feature is particularly useful in scenarios where certain users, such as customer support representatives or third-party applications, need access to specific data without exposing the full sensitive details. DDM adds an additional layer of security by dynamically masking data based on user roles and permissions, minimizing the risk of inadvertent exposure.

In a customer support scenario, agents may need access to customer data for assistance, but sensitive information like credit card numbers should be masked to comply with data protection regulations. In SQL Server 2022 or Azure SQL Database, Dynamic Data Masking can be applied to mask sensitive columns. For example, masking the “CreditCardNumber” column ensures that only authorized users, such as administrators, can view the complete credit card information, while customer support agents see only the last four digits.

```
-- Apply Dynamic Data Masking
ALTER TABLE Customers
ALTER COLUMN CreditCardNumber ADD MASKED WITH (FUNCTION
= ‘partial(0, “XXXX-XXXX-XXXX-”, 4)’);
```

When executed correctly, this statement will mask the CreditCardNumber column such that only the last 4 digits are visible, and the rest are replaced with “XXXX-XXXX-XXXX-”. For example, a credit card number like 1234-5678-9012-3456 will be displayed as XXXX-XXXX-XXXX-3456 to users who do not have the necessary permissions to view the full data.

Row-Level Security

Row-Level Security (RLS) is a feature in SQL Server and Azure SQL Database that allows you to control access to rows in a database table based on the characteristics of the user executing a query. With RLS, you can define security predicates that determine which rows a user can access, ensuring that users only see and modify the data that they are authorized to access. RLS provides a granular level of access control, enhancing data security in multi-tenant applications and scenarios where different users have different levels of data access.

Key Components of Row-Level Security:

Security Predicate: A filter predicate applied to a table that determines which rows are visible to a user. It is defined based on user attributes, such as a user ID or role.

Inline Table-Valued Function (TVF): A user-defined function that returns a table with rows filtered based on the security predicate. The function is

referenced in the RLS policy.

Security Policy: A set of rules and filters that specify how RLS is applied to a table. A security policy is created using CREATE SECURITY POLICY statements.

Use Case Scenarios for Row-Level Security:

Multi-Tenant Applications: A SaaS (Software as a Service) application serves multiple tenants on the same database. Each tenant's data must be isolated, and users should only access the data associated with their tenant. RLS is applied to tables containing tenant-specific data. The security policy ensures that users can only access rows where the TenantID column matches their assigned tenant ID.

```
-- Example: Creating a security policy for multi-tenant scenario
CREATE SECURITY POLICY TenantAccessPolicy
ADD FILTER PREDICATE dbo.fn_TenantAccessPredicate(TenantID)
ON dbo.YourTable WITH (STATE = ON);
```

Data Sensitivity: A human resources database contains sensitive information, and HR personnel should only see data related to their department. RLS is applied to HR-related tables. The security policy filters rows based on the user's department, ensuring that each HR user can only access data relevant to their department.

```
-- Example: Creating a security policy for HR department
CREATE SECURITY POLICY HRDepartmentPolicy
ADD FILTER PREDICATE dbo.fn_HRDepartmentPredicate(DepartmentID)
ON dbo.EmployeeData WITH (STATE = ON);
```

Compliance with Privacy Regulations: An application handles customer data, and there are legal requirements to ensure that only authorized personnel can access specific customer information. RLS is applied to tables containing customer data. The security policy filters rows based on the user's role and permissions, ensuring compliance with privacy regulations.

```
-- Example: Creating a security policy for customer data
CREATE SECURITY POLICY CustomerDataPolicy
ADD FILTER PREDICATE dbo.fn_CustomerDataPredicate(UserRole)
ON dbo.CustomerInformation WITH (STATE = ON);
```

Steps to Implement Row-Level Security:

Create a Security Predicate Function: Define a function that returns a predicate based on user attributes. This function is used in the security policy.

```
-- Example: Creating a security predicate function
CREATE FUNCTION dbo.fn_TenantAccessPredicate(@TenantID INT)
RETURNS TABLE
WITH SCHEMABINDING AS
RETURN SELECT 1 AS access_result WHERE USER_NAME() =
@TenantID;
```

Create a Security Policy: Use CREATE SECURITY POLICY statements to define the security policy for a specific table.

```
-- Example: Creating a security policy
CREATE SECURITY POLICY TenantAccessPolicy
```

ADD FILTER PREDICATE dbo.fn_TenantAccessPredicate(TenantID)

ON dbo.YourTable WITH (STATE = ON);

Test the Security Policy: Ensure that the security policy works as intended by testing it with various user roles and attributes.

-- Example: Testing the security policy
EXECUTE AS USER = ‘’;
SELECT * FROM dbo.YourTable; REVERT;

Manage Security Policies: Use ALTER SECURITY POLICY and DROP SECURITY POLICY statements to modify or remove security policies as needed.

-- Example: Altering a security policy
ALTER SECURITY POLICY TenantAccessPolicy
ADD FILTER PREDICATE dbo.fn_NewPredicate(TenantID)
ON dbo.YourTable WITH (STATE = ON);

Considerations and Best Practices for Row-Level Security:

Performance Impact:

Carefully design security predicates to minimize performance impact. Indexing columns used in predicates can enhance query performance.

Security Policy Granularity:

Define security policies at an appropriate granularity level. Too many policies may impact performance, while too few may compromise security.

Testing and Auditing:

Thoroughly test security policies to ensure they work as intended. Regularly audit and review policies to adapt to changing security requirements.

Dynamic Data Masking (DDM) Integration:

Consider combining RLS with Dynamic Data Masking for additional protection, ensuring that unauthorized users cannot see sensitive data even in query results.

Row-Level Security provides a powerful mechanism for fine-grained access control in SQL Server and Azure SQL Database. By defining security policies based on user attributes, organizations can enforce data access restrictions, meet regulatory requirements, and enhance overall data security.

Obfuscation of Data – SQL Server Encryption

In the dynamic landscape of data security, encryption stands as a crucial mechanism for protecting sensitive information. SQL Server 2022 and Azure SQL Database offer robust encryption options, ensuring the confidentiality and integrity of data. Encryption transforms plaintext data into ciphertext, rendering it unreadable without the appropriate decryption key. Data encryption is fundamental to securing sensitive information against unauthorized access, both at rest and in transit. In SQL Server 2022 and Azure SQL Database, encryption safeguards against potential threats such as unauthorized database access, theft of physical storage media, and interception of data during transmission. Encryption is not only a best practice but often a requirement to comply with data protection regulations.

Transparent Data Encryption (TDE)

Transparent Data Encryption (TDE) is a critical security feature available in SQL Server 2022 and Azure SQL Database. TDE operates at a file level, encrypting the entire database, including data files, log files, and backups, transparently to applications and users. The goal of TDE is to protect data at rest, ensuring that even if physical media is compromised, the data remains encrypted and inaccessible without the appropriate decryption key.

When TDE is enabled for a database, SQL Server 2022 and Azure SQL Database use a Database Encryption Key (DEK) to encrypt the actual data. This DEK is then protected by a Certificate, an Asymmetric Key, or a Hardware Security Module (HSM). The encrypted database is decrypted in memory when accessed by authorized users, applications, or services.

Protecting Sensitive Data at Rest

TDE is especially useful when databases store sensitive information such as personal identifiers, financial data, or intellectual property. By encrypting the entire database, TDE mitigates the risk of unauthorized access to the data when it’s stored on disk.

Meeting Compliance Requirements

Many compliance standards and regulations require organizations to encrypt sensitive data at rest. TDE assists in meeting these requirements, providing a comprehensive and transparent encryption solution.

Securing Backup Files

TDE extends its protection to backup files. When a database backup is created, the backup file is also encrypted. This ensures that even if backup files are misplaced or fall into the wrong hands, the data remains secure.

Performance Implications

While TDE provides robust security, it can introduce some performance overhead due to the encryption and decryption processes. Organizations should evaluate the impact on their specific workloads.

Key Management

Proper key management is essential for TDE. Regularly back up and protect the Database Encryption Key and consider using Hardware Security Modules (HSMs) for enhanced key protection.

Migration Considerations

When migrating a database with TDE enabled, ensure that the destination environment supports TDE, and transfer the necessary keys and certificates. Migrating a Transparent Data Encryption (TDE) enabled database from one SQL Server instance to another requires following steps:

Backup the Certificate and Private Key:

Before migration, back up the TDE certificate and its associated private key from the source SQL Server.

Use the following T-SQL command to create a backup:

```
BACKUP CERTIFICATE MyTDECert TO FILE =
'C:\Backup\MyTDECert.bak'

WITH PRIVATE KEY (FILE = 'C:\Backup\MyTDECertPrivateKey.bak',
ENCRYPTION BY PASSWORD = 'StrongPassword');
```

Transfer the Certificate and Private Key:

Securely transfer the certificate backup files and to the target SQL Server.

Restore the Certificate and Private Key on the Target Server:

On the target SQL Server, restore the certificate and private key using the following T-SQL command:

```
CREATE CERTIFICATE MyTDECert
FROM FILE = 'C:\Backup\MyTDECert.bak'
WITH PRIVATE KEY (FILE = 'C:\Backup\MyTDECertPrivateKey.bak',
DECRYPTION BY PASSWORD = 'StrongPassword');
```

Enable TDE on the Target Database:

Once the certificate is restored, enable TDE on the target database:

```
ALTER DATABASE MyDatabase SET ENCRYPTION ON;
```

Verify TDE Status:

Confirm that TDE is active on the target database:

SELECT name, is_encrypted FROM sys.databases WHERE name = ‘MyDatabase’;

Backup and Restore the Database:

Perform a full backup of the source database and restore it on the target SQL Server.

The restored database will retain TDE encryption.

TDE on Azure SQL Database

By default, Transparent Data Encryption (TDE) is automatically enabled for all newly deployed Azure SQL Databases. However, for older databases in Azure SQL Database, manual activation of TDE is required. TDE operates by performing real-time I/O encryption and decryption at the page level. When data is read into memory, each page is decrypted, and before being written to disk, it is encrypted again. The entire database storage is secured through symmetric key encryption, which utilizes a key known as the Database Encryption Key (DEK). Upon database startup, the encrypted DEK is decrypted and used for both database file decryption and re-encryption within the SQL Server database engine process. The TDE protector safeguards the DEK. It can either be a service-managed certificate (for service-managed transparent data encryption) or an asymmetric key stored in Azure Key Vault (for customer-managed transparent data encryption).

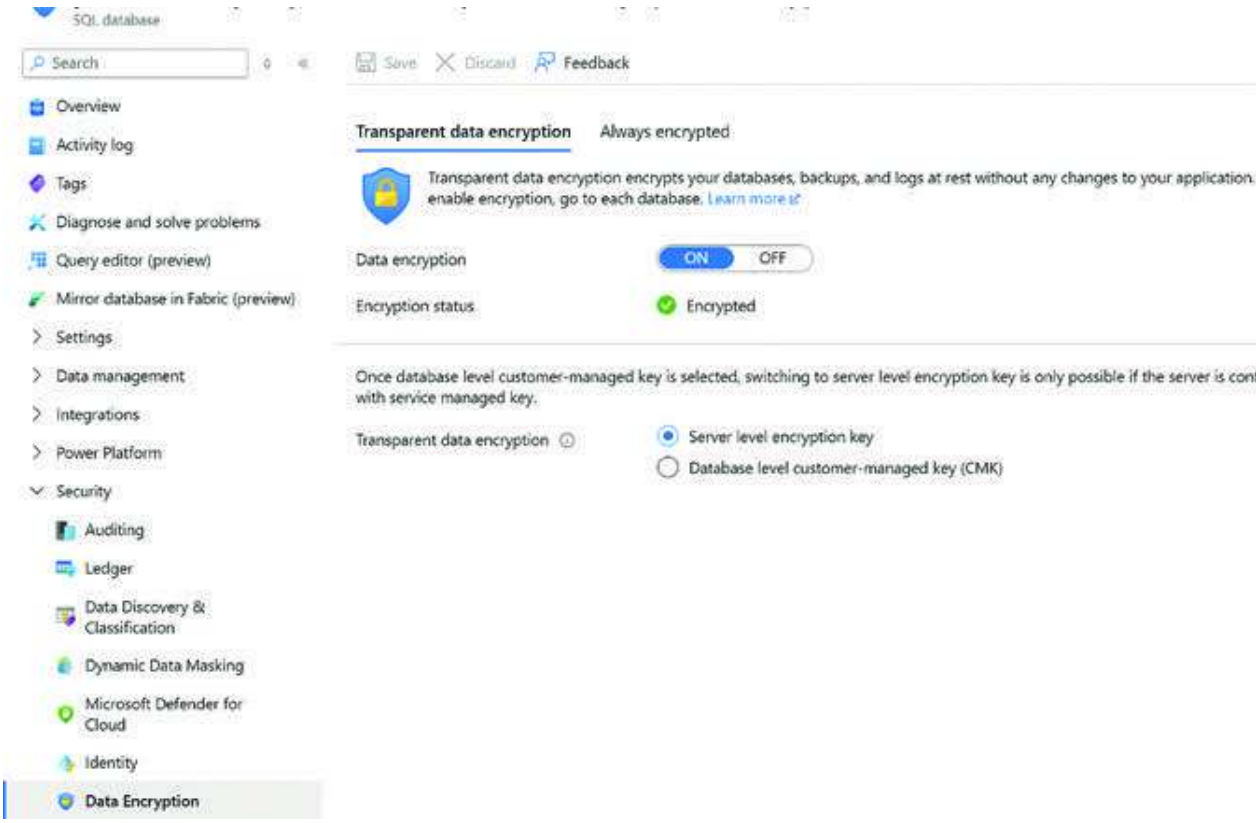


Figure 4.5: TDE on Azure SQL Database

For Azure SQL Database, the Transparent Data Encryption (TDE) protector is configured at the server level and is inherited by all databases associated with that server. By default, in Azure, TDE uses a built-in server certificate to protect the Database Encryption Key (DEK). Here are some key points about TDE in Azure:

The built-in server certificate is unique for each server, and it employs the AES 256 encryption algorithm.

In a geo-replication relationship, both the primary database and the geo-secondary databases are secured using the parent server key of the primary database.

If multiple databases are connected to the same server, they share the same built-in certificate.

This deployment model is known as Service Managed Transparent Data Encryption.

However, there is an alternative approach called customer-managed TDE (also known as Bring Your Own Key (BYOK)). In this scenario:

The TDE Protector responsible for encrypting the DEK is a customer-managed asymmetric key.

This key is stored in a customer-owned and managed Azure Key Vault, ensuring that it never leaves the key vault.

Transparent Data Encryption in SQL Server 2022 and Azure SQL Database is a powerful security feature that safeguards data at rest without requiring significant changes to the application layer. It is particularly valuable for scenarios where compliance mandates encryption or when organizations need an additional layer of defense against unauthorized access to stored data.

Always Encrypted

Always Encrypted is a feature in SQL Server and Azure SQL Database that provides an additional layer of security for sensitive data by ensuring that it remains encrypted both at rest and in transit. Unlike other encryption methods, Always Encrypted allows data to be encrypted and decrypted on the client side, ensuring that the database engine never sees the plaintext data. This client-side encryption and decryption are performed within the application, enhancing the security of sensitive data.

Key Components of Always Encrypted:

Column Master Key (CMK):

A key stored in an external key store (such as Azure Key Vault) that protects one or more Column Encryption Keys (CEKs).

Column Encryption Key (CEK):

A key used to encrypt data in a specific database column. Multiple CEKs can be protected by a single CMK.

Encryption Type:

Deterministic Encryption: Ensures that the same plaintext value is always encrypted to the same ciphertext, allowing for equality searches and joins.

Randomized Encryption: Generates a different ciphertext for the same plaintext each time, providing stronger security but limiting search capabilities.

Use Case Scenarios for Always Encrypted:

Securely Storing Credit Card Information:

Scenario: A financial application needs to store credit card information securely. Always Encrypted is used to encrypt the credit card numbers in the database, ensuring that the sensitive information remains confidential, even if there is unauthorized access to the database.

Implementation: The CreditCardNumber column is encrypted using Always Encrypted with a deterministic encryption type. The application encrypts and decrypts the data using the Column Encryption Key (CEK) stored in Azure Key Vault.

```
-- Example: Creating an encrypted column
ALTER TABLE Customers
ADD CreditCardNumber_encrypted varbinary(256)
ENCRYPTED WITH (COLUMN_ENCRYPTION_KEY = CEK_Auto1,
ENCRYPTION_TYPE = Deterministic,
ALGORITHM = ‘AEAD_AES_256_CBC_HMAC_SHA_256’);
```

Protecting Personally Identifiable Information (PII):

Scenario: An application stores personally identifiable information (PII), such as names and addresses. Always Encrypted is used to encrypt these columns, ensuring that even if the database is compromised, the sensitive information is protected.

Implementation: PII columns, such as FirstName and LastName, are encrypted using Always Encrypted with deterministic or randomized encryption. The application manages the encryption keys and performs the necessary encryption and decryption operations.

```
-- Example: Creating an encrypted column with randomized encryption
ALTER TABLE Customers
ADD FirstName_encrypted varbinary(256)
ENCRYPTED WITH (COLUMN_ENCRYPTION_KEY = CEK_Auto1,
ENCRYPTION_TYPE = Randomized,
ALGORITHM = 'AEAD_AES_256_CBC_HMAC_SHA_256');
```

Outsourcing Database Administration:

Scenario: A company outsources database administration to a third-party provider. Always Encrypted ensures that the database administrator cannot access sensitive data in plaintext, adding an extra layer of security and privacy.

Implementation: The application encrypts sensitive columns using Always Encrypted. The third-party administrator, even with access to the database, can only see the encrypted data. The client application handles the encryption and decryption processes.

-- Example: Creating an encrypted column with deterministic encryption
ALTER TABLE Employees
ADD Salary_encrypted varbinary(256)

ENCRYPTED WITH (COLUMN_ENCRYPTION_KEY = CEK_Auto1,
ENCRYPTION_TYPE = Deterministic,
ALGORITHM = ‘AEAD_AES_256_CBC_HMAC_SHA_256’);

Steps to Implement Always Encrypted:

Create Column Master Key (CMK): The Column Master Key (CMK) is a key-protecting key that encrypts one or more Column Encryption Keys (CEKs). The CMK is stored in a trusted key store, such as Azure Key Vault, to ensure its security.

-- Creating a Column Master Key
CREATE COLUMN MASTER KEY CEK_Auto1
WITH (
KEY_STORE_PROVIDER_NAME = ‘AZURE_KEY_VAULT’,
KEY_PATH =
‘https://.vault.azure.net/keys/YourCMKName/YourCMKVersion’
);

Create Column Encryption Key (CEK): The Column Encryption Key (CEK) is used to encrypt the actual data in the columns. The CEK is encrypted using the CMK, ensuring that only authorized users can decrypt it.

-- Creating a Column Encryption Key

```
CREATE COLUMN ENCRYPTION KEY CEK_Auto1
WITH VALUES (

COLUMN_MASTER_KEY = CEK_Auto1, ALGORITHM =
'RSA_OAEP',
ENCRYPTED_VALUE =
);
```

Encrypt Columns in Tables: This step involves altering the table schema to add encrypted columns. The columns are encrypted using the CEK created in the previous step.

```
-- Adding an encrypted column
ALTER TABLE YourTable
ADD YourEncryptedColumn varbinary(256)
ENCRYPTED WITH (COLUMN_ENCRYPTION_KEY = CEK_Auto1,
ENCRYPTION_TYPE = Deterministic,
ALGORITHM = 'AEAD_AES_256_CBC_HMAC_SHA_256');
```

Manage Keys in Application: The application needs to handle encryption and decryption operations using the keys. This typically involves using libraries or APIs provided by the database system.

```
// C# Example: Using ADO.NET to encrypt and decrypt data
using (SqlCommand cmd = new SqlCommand("SELECT
YourEncryptedColumn FROM YourTable", connection))
{
using (SqlDataReader reader = cmd.ExecuteReader())
{
while (reader.Read())
```

```
{

// Decrypting the data
var decryptedValue = reader.GetSqlBinary(0).Value;
// Perform actions with decryptedValue
}
}
}
```

Always Encrypted provides a robust solution for protecting sensitive data in SQL Server and Azure SQL Database scenarios, allowing organizations to maintain control over their encryption keys and ensuring that sensitive information remains confidential even when stored in the database.

Column-Level Encryption

Column-Level Encryption is a security feature in SQL Server that allows you to selectively encrypt specific columns containing sensitive information. Unlike Transparent Data Encryption (TDE), which encrypts the entire database, Column-Level Encryption provides a more granular approach, allowing you to choose which columns to encrypt and which to leave unencrypted.

Key Components:

Column Encryption Key (CEK): This key is used to encrypt the actual data in the target column. Each column that needs encryption has its own CEK.

Certificate or Asymmetric Key: The CEK is then encrypted using a certificate or asymmetric key, providing an additional layer of security.

Encryption Process:

When data is inserted or updated in an encrypted column, SQL Server uses the CEK to encrypt the data.

The CEK is then encrypted with a certificate or asymmetric key, and this encrypted CEK is stored in the database alongside the encrypted data.

During retrieval, the encrypted CEK is decrypted using the certificate or asymmetric key, and then the actual data is decrypted using the CEK.

Selective Protection of Sensitive Information: Imagine a Human Resources database where the “Salary” column contains sensitive salary information. By applying Column-Level Encryption to the “Salary” column, you can ensure that only authorized users with the necessary decryption key can access this sensitive data.

```
-- Create a Column Encryption Key
CREATE COLUMN ENCRYPTION KEY Salary_CEK
WITH VALUES (
COLUMN_MASTER_KEY = Salary_CMK,
ALGORITHM = ‘RSA_OAEP’,
ENCRYPTED_VALUE = 0x01700000016C0…);

-- Alter the Salary column to use encryption
ALTER TABLE Employee
ALTER COLUMN Salary ADD ENCRYPTED WITH
(COLUMN_ENCRYPTION_KEY = Salary_CEK);
```

Before encryption, the column data will look like this:

EmployeeID	Name	Salary
1	John Doe	50000
2	Jane Smith	60000

Figure 4.6: Table data before encryption

After encryption, the column data will look like this:

EmployeeID	Name	Salary
1	John Doe	0x01D3A5B6C7D8E9F0A1B2C3D4E5F60708090A1B2C3D4E5F60708090A1B2C3D4E
2	Jane Smith	0x02E4B6C7D8E9F0A1B2C3D4E5F60708090A1B2C3D4E5F60708090A1B2C3D4E5F6

Figure 4.7: Table data after encryption

Compliance with Data Protection Regulations: In a financial application storing credit card information, compliance with data protection regulations is crucial. By applying Column-Level Encryption to the “CreditCardNumber” column, you can comply with regulations such as PCI DSS, ensuring that only authorized entities can access and decrypt credit card information.

```
-- Create a Column Encryption Key
CREATE COLUMN ENCRYPTION KEY CreditCard_CEK
WITH VALUES (
COLUMN_MASTER_KEY = CreditCard_CMK,
ALGORITHM = ‘RSA_OAEP’,
ENCRYPTED_VALUE = 0x01700000016C0…);
```

```
-- Alter the CreditCardNumber column to use encryption
ALTER TABLE Customers
ALTER COLUMN CreditCardNumber ADD ENCRYPTED WITH
(COLUMN_ENCRYPTION_KEY = CreditCard_CEK);
```

Balancing Security and Performance: In scenarios where all columns do not require the same level of protection, Column-Level Encryption allows you to selectively encrypt only the most sensitive columns. This helps balance security requirements without introducing unnecessary overhead. Consider a customer database where the “Address” column contains non-sensitive information, while the “SocialSecurityNumber” column contains highly sensitive data. By applying Column-Level Encryption only to the

“SocialSecurityNumber” column, you optimize security measures where they are most needed.

```
-- Create a Column Encryption Key
CREATE COLUMN ENCRYPTION KEY SSN_CEK
WITH VALUES (
COLUMN_MASTER_KEY = SSN_CMK,
ALGORITHM = ‘RSA_OAEP’,
ENCRYPTED_VALUE = 0x01700000016C0…);

-- Alter the SocialSecurityNumber column to use encryption

ALTER TABLE Employees
ALTER COLUMN SocialSecurityNumber ADD ENCRYPTED WITH
(COLUMN_ENCRYPTION_KEY = SSN_CEK);
```

Column-Level Encryption in SQL Server provides a versatile and targeted approach to securing sensitive data within specific columns. By understanding how it works and identifying relevant use cases, organizations can implement a customized security strategy that aligns with their specific requirements and compliance standards.

Key Management in SQL Server: Overview and Integration with Hardware Security Modules (HSMs)

Key management is a critical aspect of maintaining the security of encrypted data in SQL Server. It involves the creation, storage, protection, and rotation of encryption keys used in various encryption features. Proper key management ensures that only authorized entities can access and decrypt the protected data.

Types of Keys:

Column Encryption Key (CEK): Used in Column-Level Encryption, encrypts the actual data in a specific column.

Database Encryption Key (DEK): Used in Transparent Data Encryption (TDE), encrypts the entire database at rest.

Master Key: Protects other keys and is often used to encrypt DEKs or CEKs.

Key Management Tasks:

Generation: Create strong, random keys using secure algorithms.

Safeguard keys from unauthorized access. Store them in a secure location, such as a key store or Hardware Security Module (HSM).

Rotation: Regularly change keys to minimize the impact of a potential compromise.

Access Control: Restrict access to keys only to authorized users or processes.

Integration with Hardware Security Modules (HSMs) for Enhanced Key Protection:

Hardware Security Modules (HSMs) are physical devices designed to provide a secure environment for key management and cryptographic operations. They offer enhanced protection against both physical and logical attacks, making them a robust choice for organizations with high-security requirements.

How HSM Integration Works:

HSMs store and manage cryptographic keys securely within a dedicated hardware device.

SQL Server, when configured to use HSMs, offloads key operations to the HSM.

Key operations, such as encryption and decryption, are performed within the HSM, providing an additional layer of protection.

Use Cases and Scenarios:

Financial Transactions: In a financial application handling high-value transactions, HSM integration ensures that encryption keys used in securing transaction data are stored and processed in a tamper-resistant environment. This helps protect sensitive financial information from potential attacks.

Healthcare Data: Healthcare databases containing patient records often require a high level of security. HSMs provide a secure enclave for key management, ensuring that patient data remains confidential and meets regulatory compliance standards.

Government Agencies: Government databases storing classified or sensitive information can benefit from HSM integration. The physical security measures of HSMs add an extra layer of protection against physical tampering or theft.

Integration Steps:

Acquire an HSM: Procure an HSM from a reputable vendor, ensuring that it meets your organization’s security and compliance requirements.

Install HSM Software: Follow the vendor’s instructions to install and configure the HSM software on the SQL Server machine.

Configure SQL Server for HSM: In SQL Server, configure the integration with the HSM by specifying the HSM provider and connection details.

Generate Keys in HSM: Use the HSM to generate and store encryption keys. SQL Server will interact with the HSM for key-related operations.

Update Key Management Processes: Adapt key management processes to leverage the capabilities of the HSM, ensuring that key generation, storage, and rotation align with HSM best practices.

```
-- Create an HSM-protected Database Encryption Key (DEK)
CREATE DATABASE ENCRYPTION KEY
WITH ALGORITHM = AES_256

ENCRYPTION BY SERVER
HSM('HSM_Provider_Name');

-- Create an HSM-protected Column Encryption Key (CEK)
CREATE COLUMN ENCRYPTION KEY
WITH VALUES (
COLUMN_MASTER_KEY = ColumnMasterKey_Name,
ALGORITHM = 'RSA_OAEP',
ENCRYPTED_VALUE = 0x01700000016C0…)
ENCRYPTION BY SERVER
HSM('HSM_Provider_Name');
```

By integrating SQL Server with Hardware Security Modules (HSMs), organizations can elevate the security of their encrypted data to meet the stringent requirements of high-security environments. Whether handling financial transactions, healthcare records, or sensitive government data, HSMs provide a reliable solution for safeguarding cryptographic keys and ensuring the integrity of encrypted information.

Securing Data Over the Wire - Connection Security in SQL Server

As data traverses networks, ensuring its confidentiality and integrity is paramount. Encryption for network protocols forms a crucial barrier against potential interception or tampering, enhancing the security of data transmissions in SQL Server environments.

Encryption for Network Protocols:

Securing data over the wire is crucial to protect sensitive information during its transit between clients and the SQL Server. SQL Server provides the capability to encrypt communication using industry-standard protocols like SSL/TLS.

Introduction to SSL/TLS Encryption for Communication:

SSL (Secure Sockets Layer) and its successor TLS (Transport Layer Security) are cryptographic protocols that provide secure communication over a computer network. They establish a secure connection by encrypting the data exchanged between a client and a server, preventing eavesdropping and tampering.

Use Cases and Scenarios:

Sensitive Data Transmission: Any scenario where sensitive data is transmitted between a client application and SQL Server, such as login credentials, personal information, or financial transactions, warrants the use of SSL/TLS encryption. This ensures that the data remains confidential during transit.

Remote Access: When SQL Server is accessed remotely, for example, from a client application running on a user's device, encrypting the connection becomes essential. This is particularly relevant in cloud-based or hybrid environments where communication traverses public networks.

Configuring SQL Server to Use Encrypted Connections:

Enable SSL/TLS in SQL Server Configuration:

In SQL Server Configuration Manager, navigate to SQL Server Network Configuration > Protocols for [Instance Name] > Right Click > Properties

Enable the "Force Encryption" option, which ensures that SQL Server uses SSL/TLS for all connections.

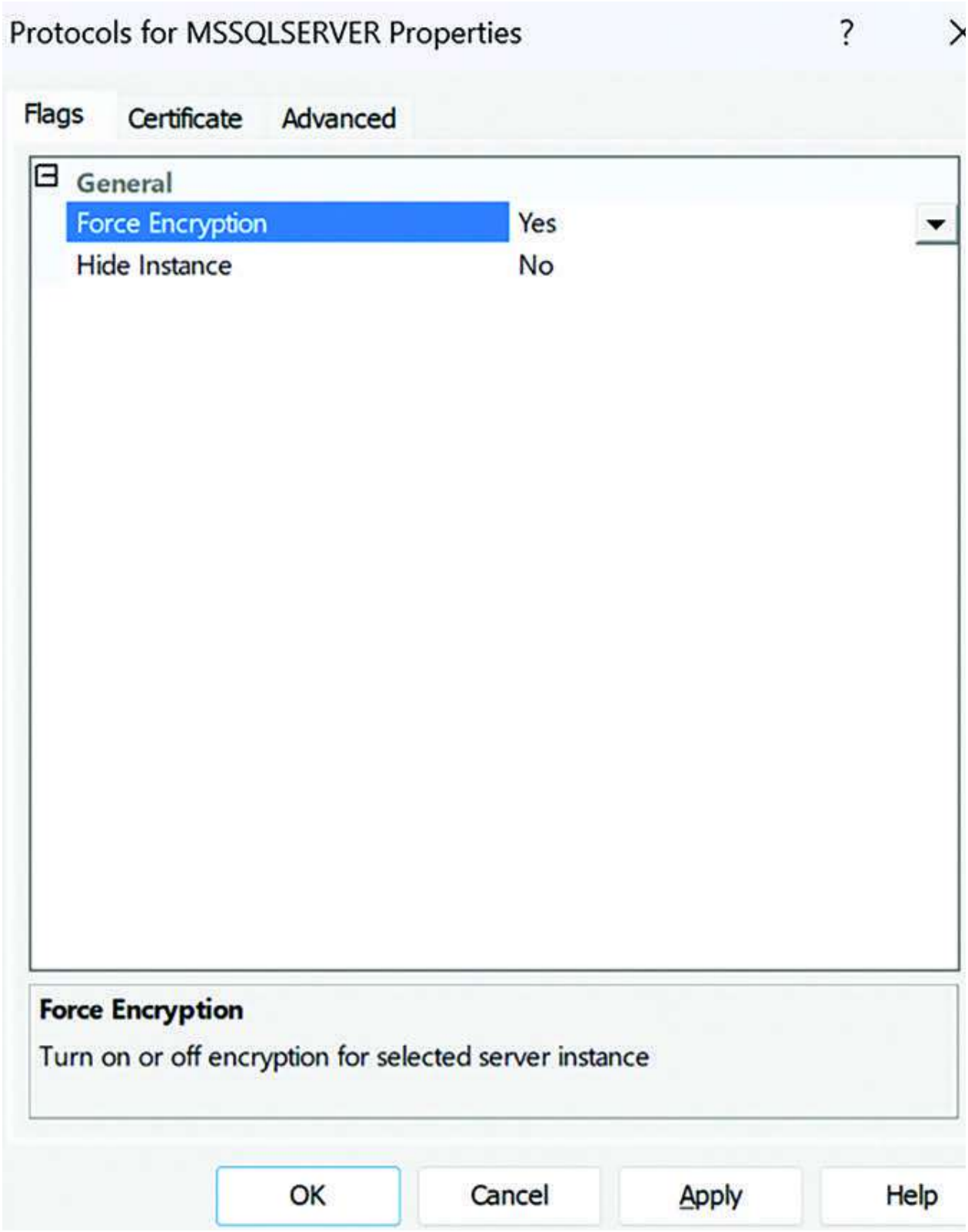


Figure 4.8: Enable SSL/TLS in SQL Server Configuration

Obtain and Install SSL/TLS Certificate:

Acquire an SSL/TLS certificate from a trusted Certificate Authority (CA) or generate a self-signed certificate for testing.

Install the certificate on the SQL Server machine.

Update SQL Server Connection String:

Modify the connection string in client applications to include “Encrypt=True” or “TrustServerCertificate=True” to enforce encrypted connections.

Data Source=mySqlServer;Initial Catalog=myDatabase; User Id=myUsername; Password=myPassword; Encrypt=True;

Connection Security Best Practices:

Firewalls and IP Filtering:

Use Case: Implementing firewalls and IP filtering helps control which computers or networks can connect to SQL Server. This is crucial for limiting access to only trusted entities, reducing the attack surface.

Scenario: In a corporate network, SQL Server is hosted on a dedicated server. Configuring a firewall to allow connections only from specific IP ranges associated with internal applications ensures that external entities cannot access the database.

-- Example IP Filtering
EXEC sp_configure ‘show advanced options’, 1;

RECONFIGURE;
EXEC sp_configure ‘ip filter allowed level’, 1;
RECONFIGURE;

Encryption of Data-in-Transit:

Use Case: Encrypting data-in-transit is essential when sensitive information, such as credit card details or medical records, is transmitted over the network. This prevents interception and unauthorized access.

Scenario: In an e-commerce application, customer transactions involve the transfer of sensitive financial data. Implementing SSL/TLS encryption ensures the confidentiality of this data during communication with the SQL Server.

-- Enforce SSL/TLS in SQL Server
ALTER ENDPOINT [TSQL Default TCP]
FOR DATA_MIRRORING (ENCRYPTION = REQUIRED ALGORITHM AES)

Regular Security Audits:

Use Case: Regular security audits help identify and address potential vulnerabilities in the network infrastructure and SQL Server configuration.

Scenario: Conducting quarterly security audits to assess the effectiveness of firewall rules, encryption protocols, and overall network security ensures ongoing protection against evolving threats.

-- Example: Check if SSL/TLS is enforced

```
SELECT name, protocol_desc, is_encryption_enabled
FROM sys.dm_exec_connections
WHERE session_id = @@SPID;
```

Recommendations for Securing the Network Infrastructure:

Keep Software and Certificates Updated: Regularly update SQL Server and the operating system. Ensure that SSL/TLS certificates are up-to-date to address vulnerabilities and maintain security.

Use Strong Authentication Mechanisms: Implement strong authentication methods, such as multi-factor authentication, to enhance the security of SQL Server connections.

Regularly Monitor and Review Logs: Monitor SQL Server logs and network logs for any suspicious activity. Regularly review logs to identify potential security incidents.

Implement Role-Based Access Control: Enforce the principle of least privilege by granting users and applications only the permissions necessary for their tasks.

Educate Users and Administrators: Provide training and awareness programs to educate users and administrators about best practices for secure network communication and potential security threats.

Securing Data Over the Wire - Connection Security for Azure SQL Database

In the digital age, safeguarding data during transmission is essential to protect against unauthorized access and interception. Specifically tailored for Azure SQL Database, connection security measures are vital to ensure the confidentiality and integrity of data flowing across networks.

Encryption for Network Protocols:

Azure SQL Database employs industry-standard security protocols to ensure the confidentiality and integrity of data during transmission over the network. The primary protocol used is TLS (Transport Layer Security), which is the successor to SSL (Secure Sockets Layer). TLS encrypts the communication channel between clients and Azure SQL Database, preventing eavesdropping and tampering.

Introduction to SSL/TLS Encryption for Communication:

Use Cases and Scenarios:

Sensitive Data Transmission: Any scenario where sensitive data is transmitted between a client application and Azure SQL Database, such as login credentials, personal information, or financial transactions, benefits from the use of SSL/TLS encryption. This ensures that the data remains confidential during transit.

Hybrid Environments: In hybrid cloud scenarios where on-premises applications communicate with Azure SQL Database over the internet, SSL/TLS encryption becomes essential to secure data transmitted over potentially untrusted networks.

Configuring Azure SQL Database to Use Encrypted Connections:

Azure SQL Database inherently enforces encryption for connections, and users do not need to explicitly configure it. However, it is essential to ensure that your client applications are configured to use encryption.

Connection String Configuration:

In the connection string of your client application, include "Encrypt=True" to enforce encryption.

Data Source=mySqlServer.database.windows.net;Initial Catalog=myDatabase;User Id=myUsername;Password=myPassword;Encrypt=True;

Network Security Best Practices for Azure SQL Database:

Firewalls and IP Filtering:

Use Case: Configuring Azure SQL Database firewall rules and IP filtering helps control which IP addresses can connect to your database.

Scenario: In a multi-tier application architecture hosted in Azure, you can define firewall rules to allow connections only from the virtual network

where your application resides. This restricts database access to trusted network segments.

```
-- Example: Configuring Firewall Rules
EXEC sp_set_firewall_rule N'Allow Application Network', '0.0.0.0', '255.255.255.255';
```

VNet Service Endpoints:

Use Case: By configuring Virtual Network (VNet) Service Endpoints, you can allow traffic only from specific VNets, further enhancing security.

Scenario: In a scenario where your application and Azure SQL Database are both within VNets, configuring VNet Service Endpoints ensures that communication remains within the trusted Azure network infrastructure.

```
-- Example: Configuring VNet Service Endpoints
ALTER DATABASE ScopedDB
ADD VNET_SERVICE_ENDPOINT (service_endpoint_value = 'Microsoft.Sql', subnet_id = '');
```

Private Link:

Use Case: Private Link for Azure SQL Database allows you to access your database securely over the Azure backbone network, avoiding exposure to the public internet.

Scenario: In a scenario where you have a database containing sensitive information, configuring Private Link ensures that data-in-transit remains within the Azure network, reducing exposure to potential threats.

```
-- Example: Configuring Private Link
CREATE PRIVATE ENDPOINT [mySqlServer]
FROM SERVICE LINKED_SERVICE ([mySqlServer]) TO ( ‘Sql’, ‘East US’ );
```

Threat Detection and Auditing:

Use Case: Utilize Azure SQL Database Threat Detection and Auditing features to monitor and respond to potential security threats in real time.

Scenario: Implementing Threat Detection allows you to receive alerts on suspicious activities, providing proactive security measures against potential attacks.

```
-- Example: Enabling Threat Detection
ALTER DATABASE ScopedDB
SET THREAT_DETECTION_POLICY = (NAME = ‘Default’, EMAIL_ACCOUNTS = ‘@example.com’);
```

Regular Monitoring and Logging:

Use Case: Regularly monitor Azure SQL Database logs and metrics to identify and respond to any unusual activities.

Scenario: Set up Azure Monitor and Azure Security Center to analyze logs and metrics, helping you stay informed about the security posture of your Azure SQL Database.

Recommendations for Securing the Network Infrastructure:

Regularly Review and Update Security Policies: Periodically review and update firewall rules, VNet configurations, and Private Link settings to align with evolving security requirements.

Implement Just-In-Time Access: For enhanced security, consider implementing just-in-time access policies to restrict database access to specific periods when needed.

Stay Informed About Azure Security Updates: Keep abreast of Azure security updates and best practices to incorporate the latest security features and recommendations.

Threat Detection and Response in SQL Server and Azure SQL Database

In the ever-evolving landscape of cybersecurity, proactive threat detection and rapid response capabilities are critical for safeguarding SQL Server and Azure SQL Database environments. Before delving into specific strategies, understanding an overview of threat detection and response provides a foundational understanding of the importance and methodologies involved.

Threat Detection and Response Overview:

Threat Detection and Response is a security feature in both SQL Server and Azure SQL Database that helps organizations identify and respond to potential security threats and suspicious activities. It involves continuous monitoring of database activities and the generation of alerts or notifications when anomalous behavior is detected. This proactive approach enables database administrators to take swift action to mitigate potential security risks.

Use Cases and Scenarios:

Identification of Anomalous Activities:

Use Case: A user account that typically accesses the database during regular working hours is suddenly initiating transactions in the middle of the night.

Scenario: Threat Detection in SQL Server or Azure SQL Database identifies this unusual pattern of access and triggers an alert. The database administrator receives a notification and investigates the activity. It turns out that the account was compromised, and a threat actor was attempting to exfiltrate data. The administrator takes immediate action, revokes the compromised account's access, and enhances security measures.

Unauthorized Access Attempts:

Use Case: Multiple failed login attempts are detected within a short time frame, suggesting a potential brute-force attack.

Scenario: Threat Detection in SQL Server or Azure SQL Database raises an alert for the repeated login failures. The administrator investigates the source of the unauthorized access attempts and identifies an external IP address attempting to gain unauthorized access. The administrator takes measures such as temporarily locking the affected accounts, implementing stronger authentication, and updating firewall rules to block the suspicious IP address.

Abnormal Data Access Patterns:

Use Case: An employee who normally accesses customer records within their department suddenly starts querying sensitive financial data.

Scenario: Threat Detection in Azure SQL Database identifies this deviation from the usual data access patterns and triggers an alert. The administrator receives a notification and investigates the situation. It is discovered that the user's credentials were compromised, leading to unauthorized access. The administrator takes corrective actions, such as disabling the compromised account, resetting passwords, and reinforcing access controls.

SQL Injection Attempts:

Use Case: An attacker attempts to exploit a vulnerability in a web application by injecting malicious SQL queries.

Scenario: Threat Detection in SQL Server or Azure SQL Database detects an influx of suspicious queries that indicate a potential SQL injection attack. The administrator is alerted and takes immediate action to secure the application, validate and sanitize user inputs, and update security measures. This proactive response prevents the successful exploitation of the SQL injection vulnerability.

Data Exfiltration Attempts:

Use Case: An insider threat tries to exfiltrate sensitive data by running large-scale queries to export information.

Scenario: Threat Detection in SQL Server or Azure SQL Database recognizes the abnormal data access pattern indicative of a potential data exfiltration attempt. The administrator is alerted, and an investigation

reveals that a disgruntled employee is attempting to copy sensitive customer information. The administrator intervenes, revokes the employee's access, and implements additional monitoring to prevent further unauthorized data access.

Threat Detection Configuration in SQL Server:

```
-- Enable Advanced Threat Protection (ATP)
EXEC sp_configure 'show advanced options', 1;
RECONFIGURE;
EXEC sp_configure 'advanced threat protection', 1;
RECONFIGURE;

-- Configure ATP Policies
EXEC sp_configure 'advanced threat protection', 1;
RECONFIGURE;
```

Threat Detection Configuration in Azure SQL Database:

```
-- Enable Threat Detection
ALTER DATABASE [YourDatabase]
SET THREAT_DETECTION_POLICY = (NAME = 'Default',
EMAIL_ACCOUNTS = '@example.com');

-- Configure Threat Detection Policies
ALTER DATABASE [YourDatabase]
SET THREAT_DETECTION_POLICY = (NAME = 'CustomPolicy', …)
```

Implementation in SQL Server:

In SQL Server, Threat Detection is part of the Advanced Threat Protection (ATP) feature. It includes:

Vulnerability Assessment: Conducts regular assessments to identify potential vulnerabilities and security issues in the database.

Advanced Threat Protection: Monitors database activities, detecting anomalous patterns and potential security threats.

Dynamic Data Masking: Protects sensitive data by dynamically masking it based on user roles and permissions.

Transparent Data Encryption: Encrypts data at rest, ensuring that even if unauthorized access occurs, the data remains encrypted.

Implementation in Azure SQL Database:

In Azure SQL Database, Threat Detection is a built-in feature and part of the Azure Security Center. Key components include:

Threat Detection Policies: Configurable policies allow administrators to define the conditions that trigger alerts, such as multiple login failures or data access anomalies.

Alerts and Notifications: Threat Detection generates alerts and notifications when suspicious activities are identified. These alerts can be sent via email or integrated with Azure Monitor for a centralized view.

Incident Response: Threat Detection provides recommendations and guidance on responding to security incidents. It assists administrators in taking appropriate actions to mitigate risks.

Integration with Azure Active Directory: Utilizes Azure Active Directory identities for authentication and access control, enhancing security and ensuring proper user verification.

Best Practices for Threat Detection and Response:

Regularly Review Alerts: Database administrators should routinely review Threat Detection alerts to identify and respond to potential security incidents promptly.

Fine-Tune Policies: Customize Threat Detection policies based on the specific security requirements and patterns of normal behavior in the organization.

Collaborate with Security Teams: Collaborate with broader IT security teams to share threat intelligence and coordinate responses to potential threats that might span multiple systems or services.

Continuous Monitoring: Implement continuous monitoring practices to ensure that the Threat Detection and Response mechanisms are up-to-date and effective against evolving security threats.

By leveraging Threat Detection and Response capabilities in SQL Server and Azure SQL Database, organizations can enhance their security posture, swiftly respond to potential threats, and mitigate the impact of

security incidents. These features play a crucial role in maintaining the integrity, confidentiality, and availability of data in database environments.

Here is a reference of best practices and recommendations for securing SQL Server 2022 and Azure SQL Database:

Database: Database: Database:
Database: Database:
Database: Database:
Database: Database:
Database: Database: Database: Database: Database: Database: Database: Database: Database: Database: Database: Database: Database: Database:

Database: Database: Database: Database: Database: Database: Database:

Database: Database:
Database: Database:
Database: Database:

Table 4.1: Best practices and recommendations for securing SQL Server 2022 and Azure SQL Database

Recommendations for Maintaining a Secure Azure SQL Database Environment:

Continuously Monitor Azure Security Center:

Regularly review Azure Security Center recommendations and implement suggested security enhancements.

Enable Advanced Threat Protection (ATP):

Activate ATP for Azure SQL Database to detect and respond to potential security threats.

Leverage Azure Policy and Blueprints:

Enforce security controls and compliance standards using Azure Policy and Blueprints.

Implement Just-In-Time Access:

Use Azure Just-In-Time VM Access to restrict access to Azure SQL Database when not needed.

Utilize Azure Key Vault for Key Management:

Store and manage cryptographic keys securely in Azure Key Vault for enhanced key protection

Conclusion

This chapter provided a comprehensive overview of security considerations in SQL Server and Azure SQL Database, emphasizing the importance of a multi-faceted approach to safeguard data and ensure the integrity and confidentiality of database environments. The protection of SQL Server and Azure SQL Database environments requires a comprehensive approach, involving various security measures to mitigate risks effectively. Authentication and authorization mechanisms, such as Azure Active Directory Authentication, ensure that only authorized users access the database, while features like Always Encrypted provide an additional layer of security by encrypting sensitive data both at rest and in transit.

Connection security measures, including encryption for network protocols and SSL/TLS configurations, safeguard data during transmission, while robust network security practices such as firewalls and IP filtering prevent unauthorized access. Row-level security enables fine-grained control over data access, making it ideal for scenarios like multi-tenant applications and compliance with privacy regulations. Threat detection and response features play a crucial role in identifying and mitigating security threats in real-time, helping organizations stay ahead of potential breaches. Best practices, including regular security audits, role-based access control, and continuous training, are essential for maintaining a secure environment.

Continuous learning is key in the ever-evolving landscape of cybersecurity. Leveraging resources like Microsoft Learn and Azure documentation, and actively participating in security communities, ensures that organizations stay updated on emerging threats and best practices. By implementing these strategies, organizations can effectively protect their data assets and maintain the integrity and confidentiality of their database environments.

The next chapter will discuss business continuity strategies for SQL offerings involving implementing measures such as regular backups, disaster recovery plans, and high availability solutions to minimize downtime and ensure uninterrupted access to critical data to mitigate the impact of potential disruptions and maintain operational resilience in their SQL environments.

CHAPTER 5

Business Continuity Strategies for SQL Offerings

Introduction

In the design of data estate architecture, one of the critical aspects of the design is the Business continuity strategies in case of planned maintenance, a failure that could bring down the data estate, or a disaster that could take down an entire data center. The design should incorporate plans on how these situations can be handled effectively and minimize the downtime of the critical applications.

In this chapter, we will learn about the common Business continuity strategies available with SQL Server across Windows, Linux, and Containers. In later sections of this chapter, we will have a comprehensive discussion on how business continuity strategies work with SQL Server on Linux and containers and how the strong partner-based Linux ecosystem supports these high availability strategies.

Structure

In this chapter, we will cover the following topics:

Recovery Point Objective versus Recovery Time Objective

High Availability versus Disaster Recovery

Business Continuity Strategies for SQL Server

Business Continuity Strategies for Azure SQL

Recovery Point Objective versus Recovery Time Objective

Almost in every High Availability and Disaster Recovery (HA/DR) discussion, one of the common terms that you will hear are the Recovery Point Objective (RPO) and Recovery Time Objective (RTO). Normally, the RPO and RTO are decided by the organization based on the suggestions from various teams such as Applications, System, and Data Architects. Now, let us understand these terms first and then let us look at the different HA/DR strategies.

Recovery Point Objective: It is basically a metric that helps you identify the tolerance limit for data loss for your organization measured in time, thus helping you make decisions on your backup strategies.

Here is an example: if your organization is willing to endure data loss of up to 15 minutes in the event of a disaster. Then, as part of your backup strategy, you will plan on taking full backups and log backups at a frequency of every 15 minutes, and then your RPO is 15 minutes because, with this strategy, you ensure that the organization does not lose more than 15 minutes of data, provided you have all your backup files intact with you to run the restores.

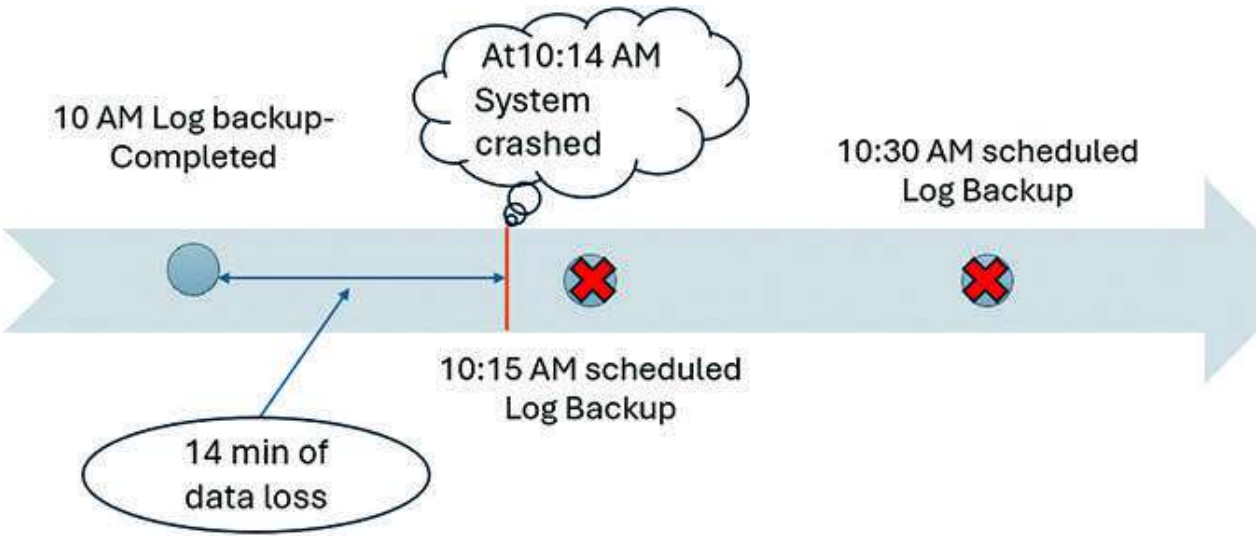


Figure 5.1: Recovery Point Objective

As shown in Figure the log backups were scheduled to happen every 15 minutes, and at 10 AM there was a log backup that occurred, and at 10:14 AM unfortunately the system crashed, and apart from backups there were no other strategies in place for business continuity, and thus you end up losing 14 minutes of data, that is, data from 10 AM to 10:14 AM (as this data was not backed up in any of the previous backups), but this still falls under the 15 minute RPO and is thus acceptable for your organization.

Recovery Time This metric focuses on how long the system can remain down after an issue without catastrophically impacting the business. In other words, the important thing is the time it takes for the system to be back online and functional after a crash or downtime has occurred before the business gets critically impacted.

To give you an example, if your organization decides on an RTO of 30 minutes, that means once the system experiences downtime or a crash, then the entire system should be back online within the next 30 minutes from the time it was down. Because any delay in bringing the system back online after

30 minutes causes a significant impact on the business, which is not acceptable to the organization. Here is a timeline example to explain this scenario.

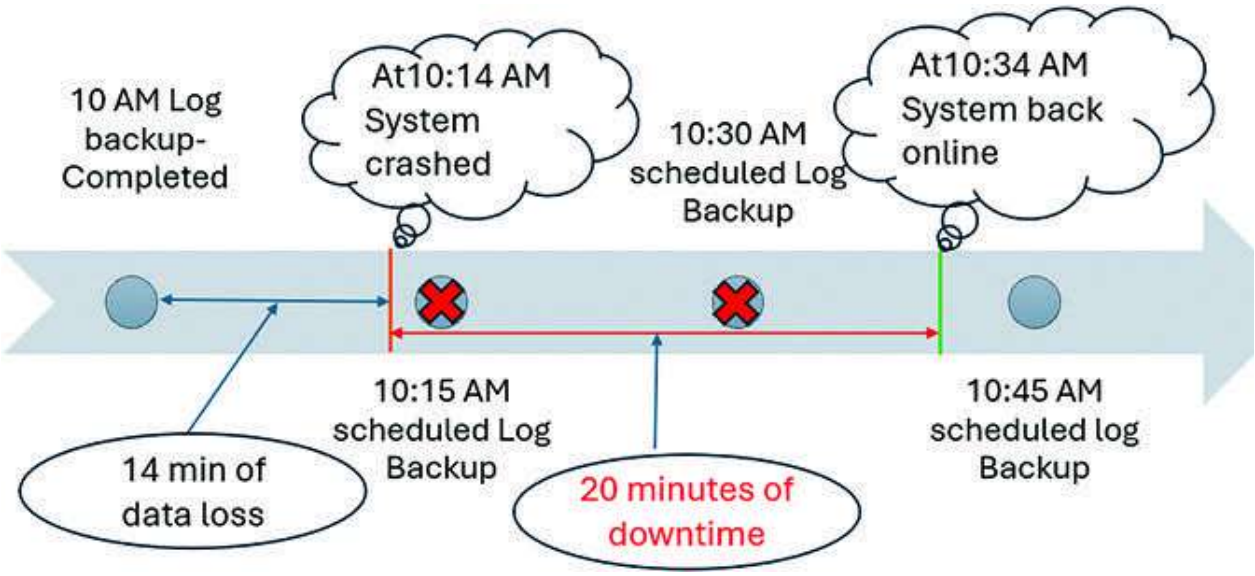


Figure 5.2: Recovery Time Objective

As shown in Figure the system crashed at 10:14 AM, and there were no backups that happened at 10:15 AM and 10:30 AM, as the system was down. The system was online and operational back again at 10:34 AM, which means the total downtime was for about 20 minutes from 10:14 AM to 10:34 AM, and since the decided RTO was 30 minutes, this is an acceptable downtime for the business as the system was up and running in less than 30 minutes.

High Availability versus Disaster Recovery

Now that you understand Recovery Point Objective (RPO) and Recovery Time Objective (RTO), it is time to look at the two more jargons mostly used with business continuity strategies, and they are High Availability (HA) and Disaster Recovery (DR). In discussions with customers and partners, these two concepts are often used together and sometimes incorrectly. It is important to address these concepts, which are related but distinct from each other.

High Availability (HA), as the name suggests, is all about ensuring that the system is highly available; in other words, how can you avoid or prevent downtime? While Disaster Recovery (DR) focuses on plans in place to recover from a major outage or downtime.

The way we can explain this to customers is to think of HA as your plan A, which is all about how you can avoid downtime. And DR is your plan B; in case plan A fails, then how fast can you recover from the outage and ensure your RTO time is not breached?

Both HA/DR are critical aspects of design and architecture for any business to ensure that it continues to operate where there is downtime, crashes, or catastrophic events such as fire, earthquake, or other natural disasters that can take down an entire data center. Also, HA/DR should be thought about not only for databases but also at the system and application level to ensure that the resilience of the entire IT infrastructure is never compromised, and it continues to function during issues and can be

brought online within required RTO and RPO specifications in case of a downtime or crash.

Business Continuity Strategies for SQL Server

In the previous section, we learnt the general concepts of HA/DR, RTO, and RPO. Now it is time to look at the different features that are available in SQL Server to enable HA/DR as part of your business continuity strategy. In this section, we will focus on all the available HA/DR features and when you need to implement each of these features or a mix of these features. Like in the previous chapters, in this chapter as well we will have a dedicated section for SQL Server on Linux and containers to ensure you are empowered to choose and deploy your choice of HA/DR technology on your choice of SQL Server deployments.

Let us get started with the HA/DR features available in SQL Server:

Log shipping

Always On Failover Cluster Instances (Always On FCI)

Always On availability Groups (Always On AG)

In the list, as you can notice, we are not talking about the Mirroring technology, as this feature was deprecated by Microsoft with the release of SQL Server 2012, with the introduction of Always On availability groups that comes with better features when compared to Mirroring.

Log shipping

One of the simplest features that provides both High Availability (HA) and cost-effective Disaster Recovery (DR) built on native SQL Server backups, such as the full and transaction log backups. There is no concept of automatic failover, thus not recommended as an HA solution for production workloads because the failover requires manual intervention. Also, this is mostly recommended to be used as a cost-effective DR option across data centers, as you can ship logs over the wire and do not have to maintain or setup disk latency or other tools and capabilities to help keep the data center storages in sync. This feature allows for a robust protection mechanism against extensive corruption. If the primary server encounters corruption, logshipping provides a window of opportunity to halt log restorations on secondary servers, enabling data recovery from these secondary servers.

Log shipping, as the name suggests, ships the logs from one server to the other. Log shipping basically involves three steps:

Backup the transaction log at the primary server.

Copy the transaction log backup to the secondary SQL Server.

Restore the copied transaction log backup on the secondary server.

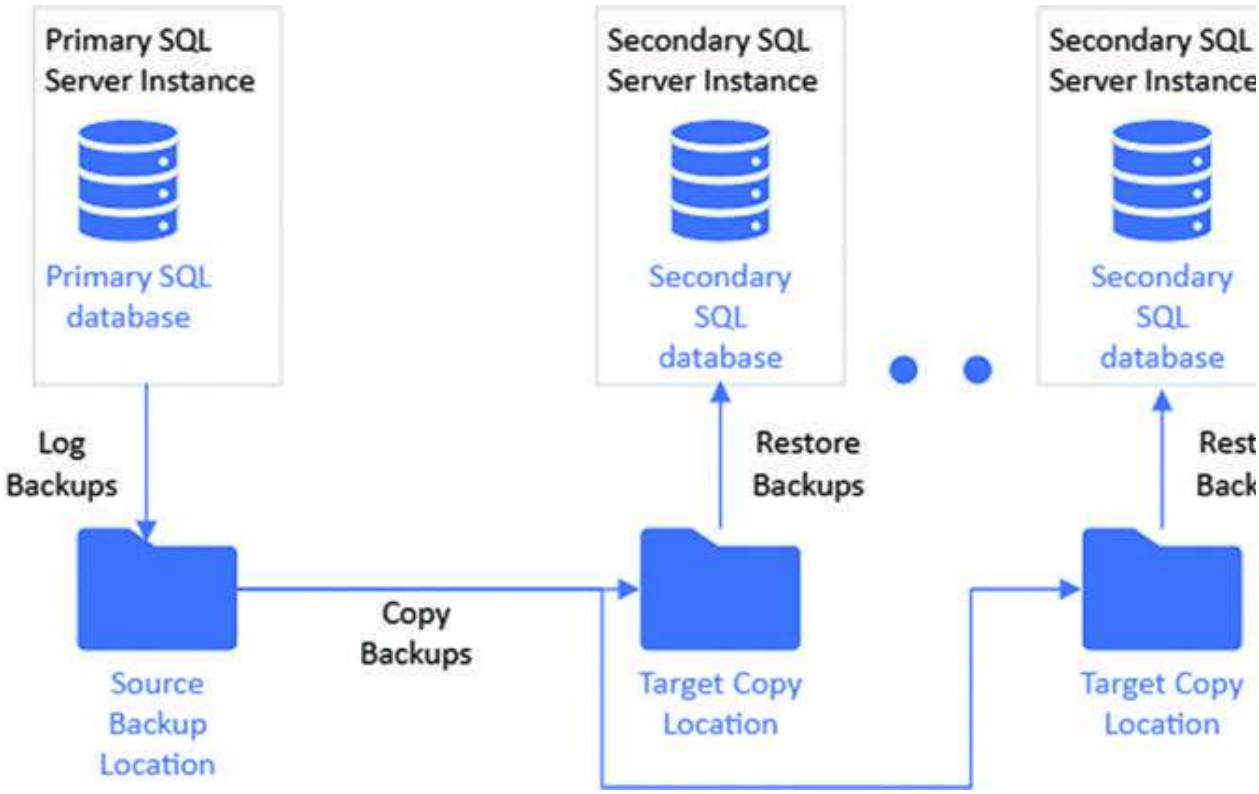


Figure 5.3: Log shipping in action

As shown in Figure all that is needed for you to setup log shipping are a couple or more instances of SQL Server, a SQL Server database from which you configure the backups to occur based on a set frequency, and then these backups are copied from the backup location to the destination location, and once the backups are copied, you restore the copied backups on the secondary SQL Server instances, thus creating the SQL Server secondary database.

As you may have noticed, you can have more than one secondary, so you can have one secondary on the same data center, and as part of your DR strategy, you can have another secondary across data center. Also, this is a database level HA/DR solution, so you do not ship the logins, users, and other instance-specific details that may be needed for a successful failover of an application from the old primary to the new primary.

You can customize the restore frequency that gives you the ability to fix issues on primary due to inadvertent human errors such as running updates without the where clause on the primary. If you face such a situation in a production environment, you can:

Stop the restore of the log backup on the secondary.

Temporarily point the application to connect to the secondary after you have manually changed the role from standby to primary.

Then once you have fixed the issue on the old primary, you can revert the application to the old primary and change the role from primary to standby for the current primary.

Giving you the ability to fix issues on the primary while ensuring there is very little downtime for the application. The downtime will primarily be the time taken to manually switch the roles of one of the secondary SQL Server instance from standby to primary. It is recommended to use Log shipping together with other HA/DR features and not solely rely on this feature as a HA/DR when designing the business continuity strategy for production workloads.

Log shipping is compatible with SQL Server on both Linux and Windows platforms. To set up database log shipping using SQL Server Management Studio (SSMS), here are the steps to follow.

Connect to the primary SQL Server and right-click the database that you want to enable log shipping for. Figure 5.4 shows the sample page that opens up.

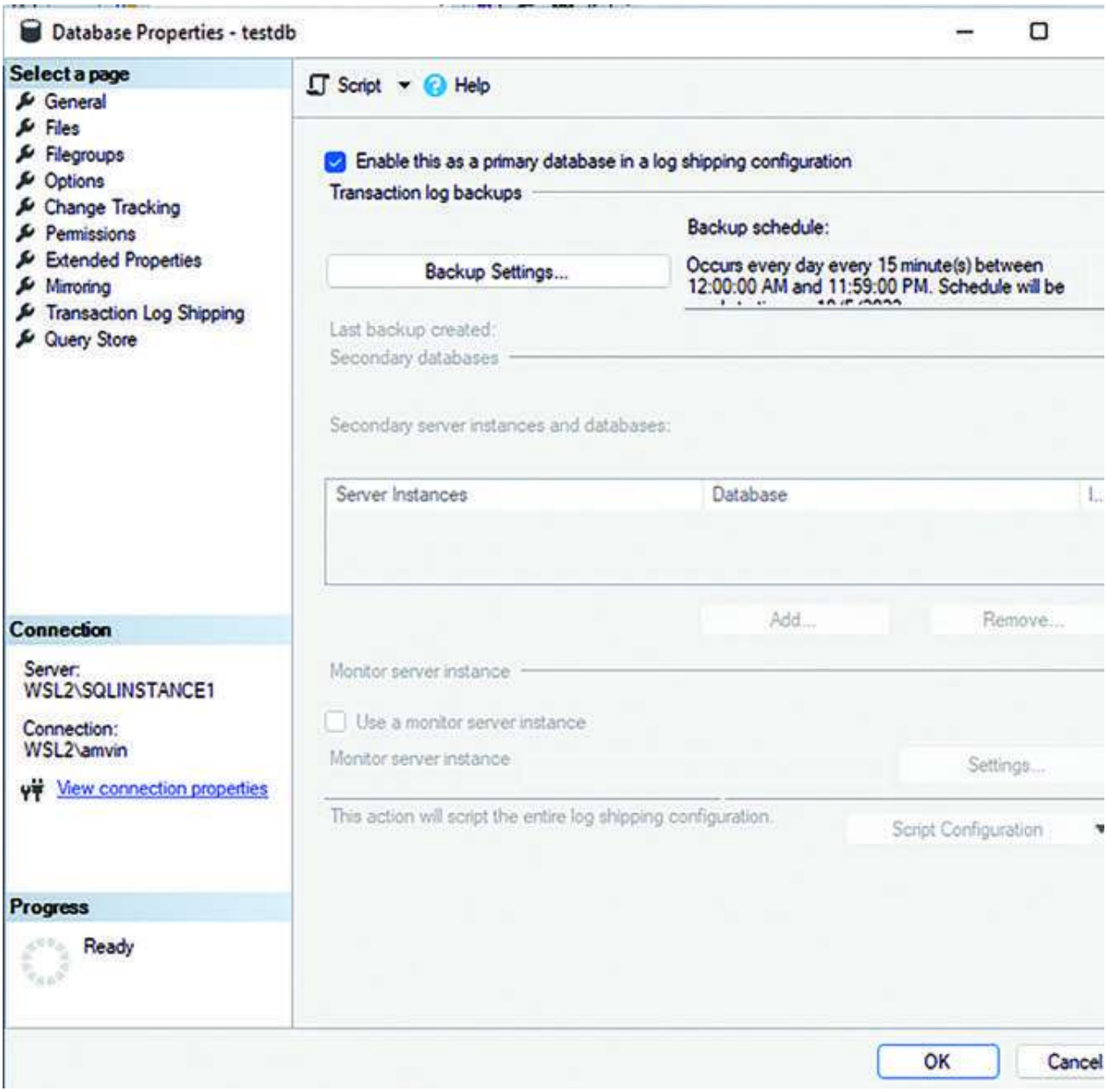


Figure 5.4: Log shipping setup screen in SSMS

As you can see, it is a simple UI. As soon as you check the Enable this as a primary database in a log shipping configuration, you can configure the backup settings as shown in Figure You must provide the backup destination and can provide the name of the backup job that gets created on the primary server. You also have the option to configure the delete option for files older.

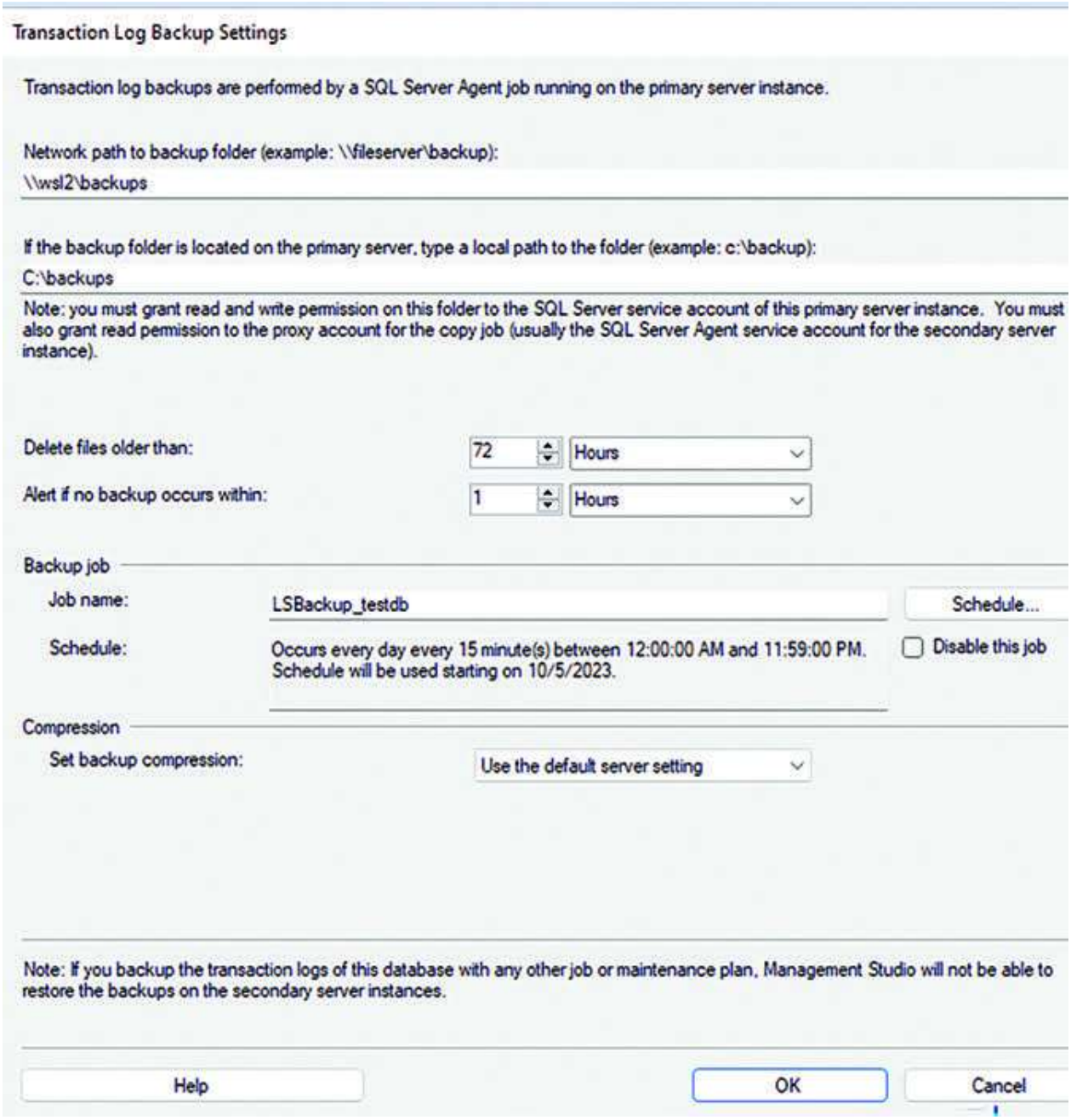


Figure 5.5: Logshipping backup setting page in SSMS

Now, you configure the Secondary details, as shown in Figure where you initialize the backup job.

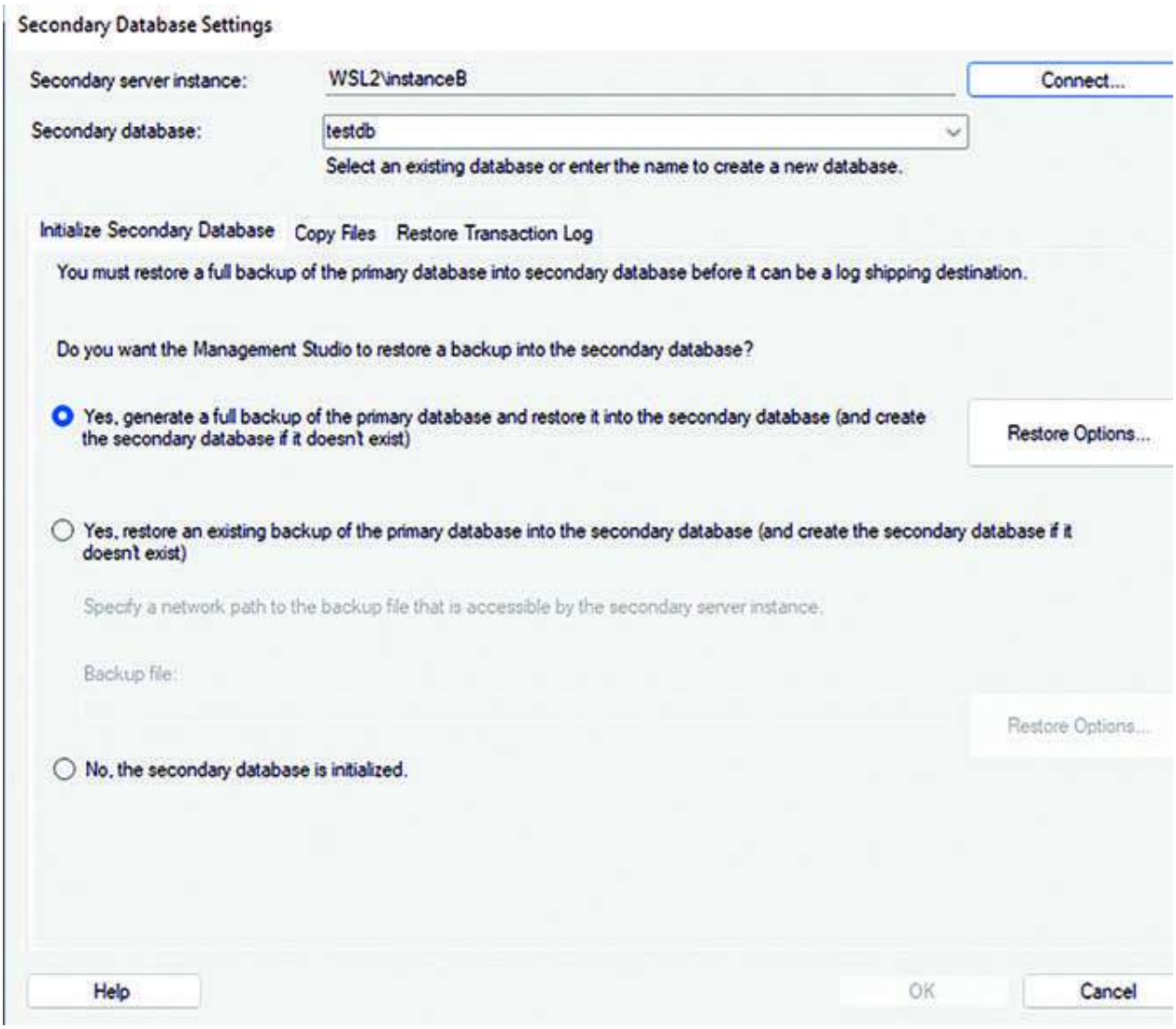


Figure 5.6: Logshipping secondary database setting in SSMS

Followed which you configure the copy job that gets created on the secondary server, as shown in Figure

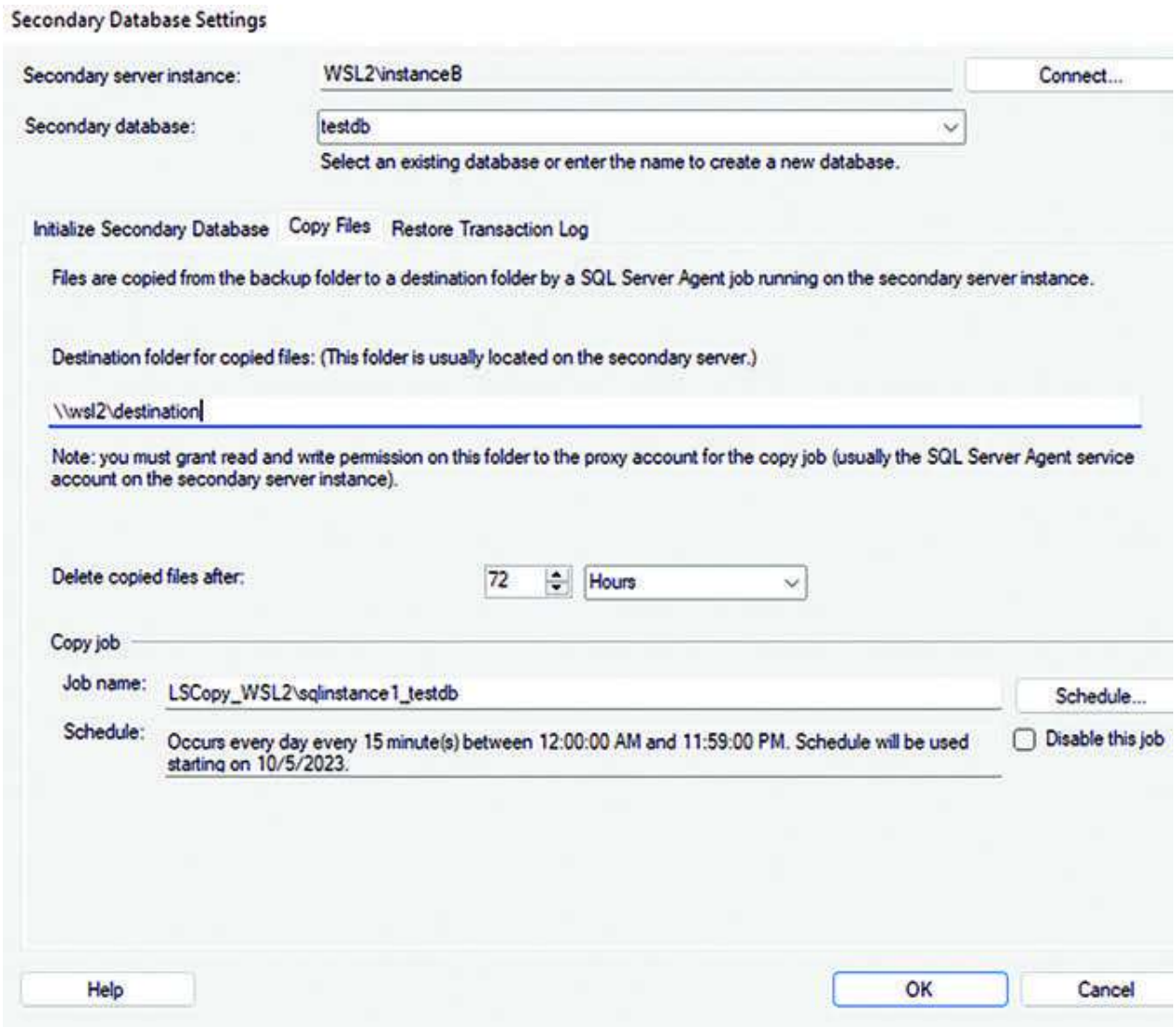


Figure 5.7: Logshipping secondary database destination folder setting in SSMS

Finally, you configure the restore job, again created on the secondary server. During the restore, as you can see, you have two options to choose from, as shown in Figure Based on your requirements, you can choose any one of the options.

No Recovery Mode: The database on the secondary is not in an accessible state.

Stand By The database is accessible for ready-only queries and can be accessed by users. This database will not be available when the restore is happening, but apart from that, it is available for access for ready-only queries.

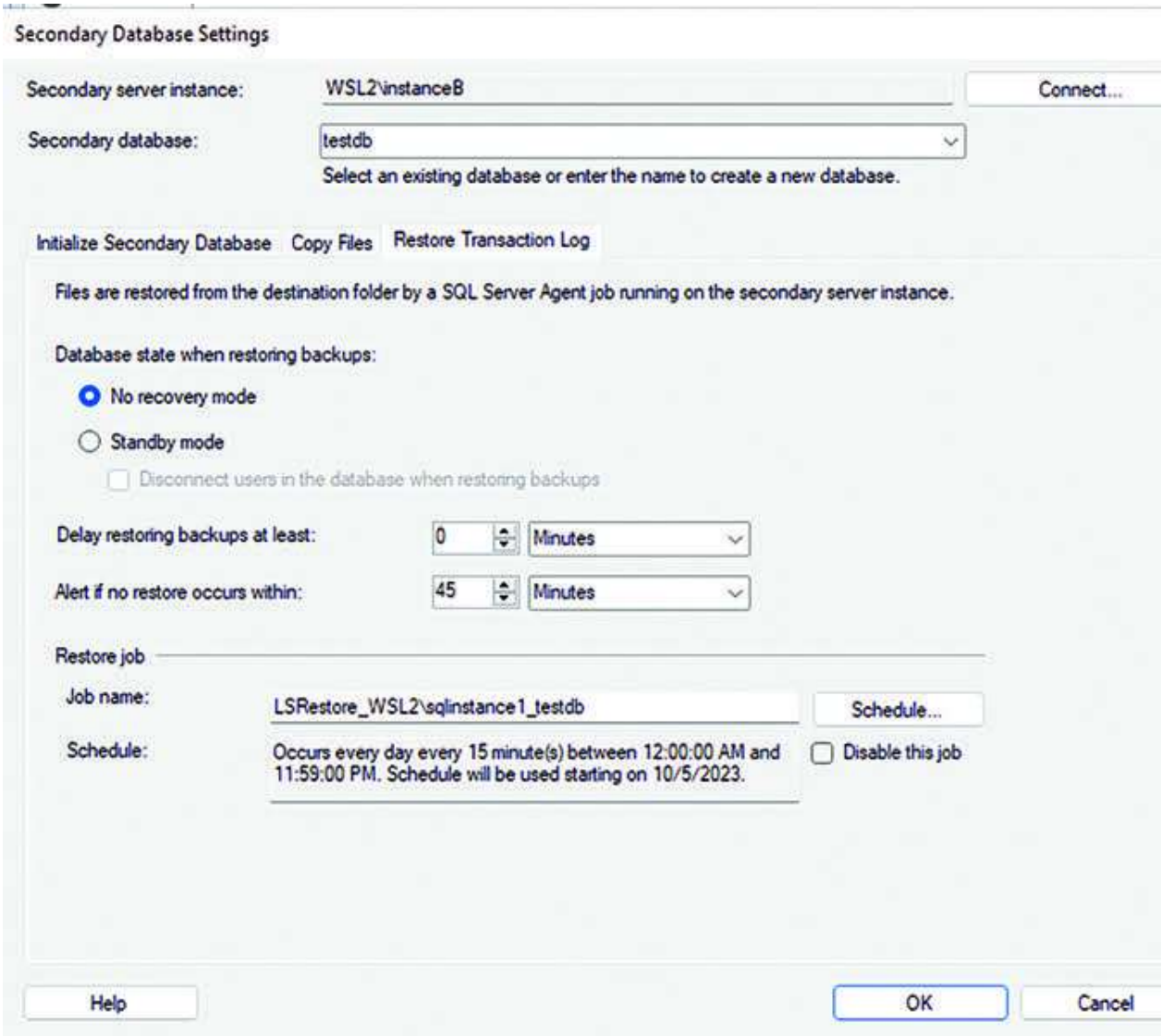


Figure 5.8: Logshipping secondary restore jobs settings in SSMS

Once the configuration is complete, you will see the jobs created on the primary and secondary servers as shown in Figure To your left are the job listings on the primary server and to the right are the job listings on the secondary server.

Figure 5.9: Listing the Logshipping related jobs created on the primary and secondary servers.

Always On in SQL Server

Prior to delving into the topic of Always On Failover Clustered Instances, it is important to clarify that Always On is a comprehensive term encompassing two high-availability/disaster recovery (HA/DR) features:

Always On Availability Group (AG)

Always On Failover Cluster Instance (FCI)

Introduced in SQL Server 2012, Always On is not merely another name for the Availability Groups (AG) feature. This distinction is crucial as the terms Always On and AG are often used interchangeably, which is not correct. The Availability Group feature was also a new HA/DR feature released with SQL Server 2012. Therefore, understanding the difference between these terms is essential for accurate communication and implementation.

Both the Always On AG and Always On FCI work for SQL Server across Windows and Linux platforms. For SQL Server containers only Always On availability group is supported for production workloads. There is no support for Always On failover Clustered instance for SQL Server deployed as containers on Linux-based distributions.

Let us understand the cluster stack and the difference when SQL Server is deployed on Windows versus Linux. The primary reason for addressing this topic is that many customers have frequently inquired about the existence of multiple clustering technologies for SQL Server on Linux. This section aims to provide a comprehensive answer to that question.

To try and answer this question, let us look at the cluster stack for SQL Server on Linux versus SQL Server on Windows. This in no way is a technical architecture breakdown, but just a simple diagram to help understand the major components and how they compare against each other when you deploy SQL Server on various platforms.

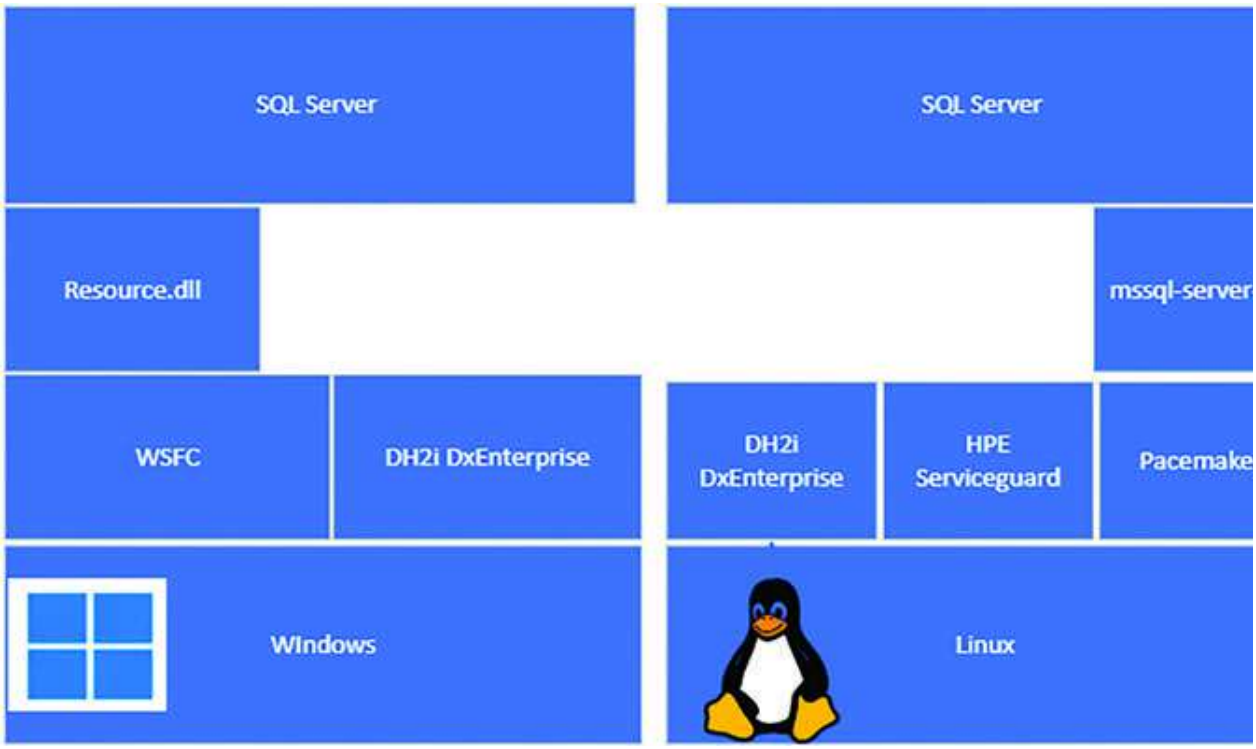


Figure 5.10: Cluster stack comparison for SQL Server on Windows versus Linux

As you can see in Figure when you deploy SQL Server on Windows, you can choose the clustering stack to be Windows Server Failover Cluster (WSFC) or you could also choose another clustering stack provided by DH2i DxEnteprise This by far is not the exhaustive list of all the other cluster stacks that are supported for SQL Server on Windows. This by no means is an exhaustive list of the cluster stacks supported by SQL Server on Windows; a few are listed here for explanation only.

Now, when you deploy SQL Server on a WSFC cluster, it uses Resource.dll, which is a separate executable that contains specific code to help manage SQL Server resource within the cluster. It basically provides WSFC with functionalities such as starting, stopping, and monitoring the SQL Server resource. Thus, the resource dll acts like a bridge between SQL Server and WSFC to ensure that WSFC can orchestrate and manage the SQL Server resources inside the cluster.

When you compare this with the Linux cluster stack, you notice there is no WSFC as a cluster technology option. That is because the WSFC is supported only on Windows and not on Linux. But, on Linux, you have an entire ecosystem of clustering technologies such as the DH2i DxEnteprise HPE Serviceguard Pacemaker and more. SQL Server on Linux today is supported on all the aforementioned clustering technologies. This basically gives you, as a customer or user, the advantage of deploying SQL Server on your choice of clustering technology.

When you choose to deploy SQL Server on Pacemaker as the clustering technology, then instead of using the resource.dll (WSFC), you now have the mssql-server-ha agent, which provides the similar functionality to resource.dll, such as starting, stopping, monitoring, and validating. The code is also available on GitHub: https://github.com/microsoft/mssql-server-ha

But other clustering technologies, such as DH2i DxEnterprise and HPE Serviceguard, do not depend on resource.dll or mssql-server-ha. They directly communicate with the SQL Server and manage that as a clustered resource for any Always On feature.

For SQL Servers deployed as containers as of writing this book, the only Always On feature supported is the Availability Groups (AGs), and the clustering stack used is DxEnterprise from DH2i. We will learn more about the AGs for SQL Server containers on Kubernetes and without Kubernetes in the later sections of this chapter.

Always On Failover Cluster Instances (FCI)

Always On Failover cluster instance is a reliable and time-proven high availability feature that provides high availability at the instance level. Hence, when a failover occurs, not only do the databases failover but also other instance-level objects such as logins, system databases, endpoints, and certificates failover; basically, the entire instance is now available on the new node.

Here are the basic steps involved in deploying a SQL Server Always on FCI on a Windows Server failover cluster (WSFC) stack:

Create and configure a Windows Server failover Cluster (WSFC) with the nodes and shared storage that you intend to use for your SQL Server FCI. For detailed steps on the creation and configuration of WSFC, please refer to the official Microsoft documentation here: Create a failover cluster | Microsoft Learn

Next, run the SQL Server setup to deploy the SQL Server Failover clustered instance on one of the cluster nodes. You need to provide the cluster disk resources, virtual network name, and IP address. For details, please refer to the official Microsoft documentation: Create new Failover Cluster Instance - SQL Server Always On | Microsoft Learn

Once you have the resource group created inside the WSFC from the previous step, you can now add nodes to the existing SQL Server cluster as described in the steps here: https://learn.microsoft.com/en-us/sql/sql-server/failover-clusters/install/add-or-remove-nodes-in-a-sql-server-failover-cluster-setup?

view=sql-server-ver16# Add This ensures that you run the SQL Server setup on each node that needs to be added to the cluster.

Figure 5.11 shows the basic representation of a SQL Server Failover Cluster Instance.

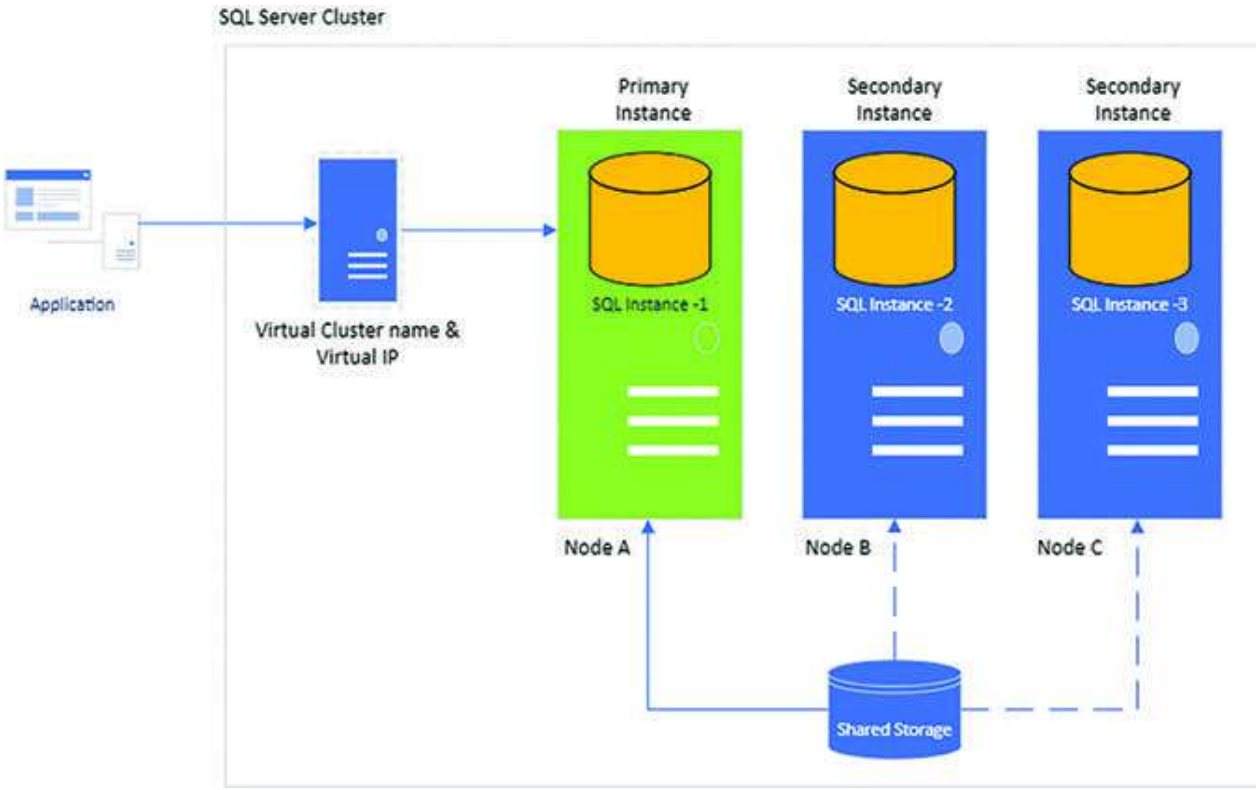


Figure 5.11: SQL Server Failover Clustered Instance with shared storage

There are three separate individual SQL Server instances running three different but identical software configurations, including the same versions of the operating system and SQL Server. As these SQL Server instances are running in clustered configuration, the SQL Server instances are managed by WSFC and thus not automatically started on their own.

Even though you see three separate SQL Server instances, they do not have their own storage, but they use the shared storage, also known as the clustered

storage, that was configured to be used during the SQL Server installation. This configuration ensures that all nodes within the Failover Cluster Instance (FCI) maintain a consistent view of the instance data in the event of a failover.

A resource group inside the Windows Server Failover cluster for SQL Server contains the minimum resources needed to run a SQL Server FCI. These resources are:

Virtual Network Name A DNS name that identifies the FCI to the client applications.

Virtual IP Address: An IP address that is assigned to the VNN and points to the current primary node in the cluster.

SQL Server: The SQL Server service that hosts the databases and provides data access.

SQL Server Agent: The SQL Server Agent service that runs scheduled jobs and tasks.

Clustered disks: The physical disks that store the SQL Server database files, log files, and other files.

Only one node can be the primary node at a time, which owns and controls these resources. The WSFC service manages these resources as a group and moves them together to another node in case of a failover. The resources also have dependencies that determine the order in which they are brought online or offline. These dependencies are essential for ensuring the availability of the SQL Server instance.

The SQL Server agent resource is dependent on the SQL Server resource, meaning that the SQL Server Agent service will start after the SQL Server service has started.

SQL Server resource is dependent on the SQL Server virtual network name (VNN) and the SQL Server clustered disks, this implies that the disks and the SQL Server VNN needs to be online before SQL Server can come online.

The SQL Server virtual network name in turn depends on the SQL Server IP address resource.

The SQL Server FCI uses a VNN and an IP address to simplify client connectivity. The VNN is a DNS name that identifies the FCI to the client applications. It is associated with a virtual IP address that points to the current primary node in the cluster. The VNN and the virtual IP are registered in Active Directory and DNS, so that clients can connect to them without knowing the actual node name. This way, the clients do not have to change their connection strings after a failover, as the VNN and the virtual IP remain the same.

Failover is the process of switching the SQL Server instance from one node to another in case of a failure or maintenance. There are mainly two types of failovers:

This happens when the cluster detects that the current node is not working and moves the SQL Server instance to another node automatically. This can reduce downtime and data loss.

This is done by the administrator who initiates the failover to perform some tasks on the current node or for testing purposes. This gives more control and

flexibility to the administrator.

During a failover, the resources are taken offline on one node and brought online on the other node according to the dependencies that are set up as described before. Figure 5.12 shows the cluster after a failover, where node B becomes the primary and the VNN and IP address point to node B. This ensures that the application can continue to work without changing the connection string after the failover.

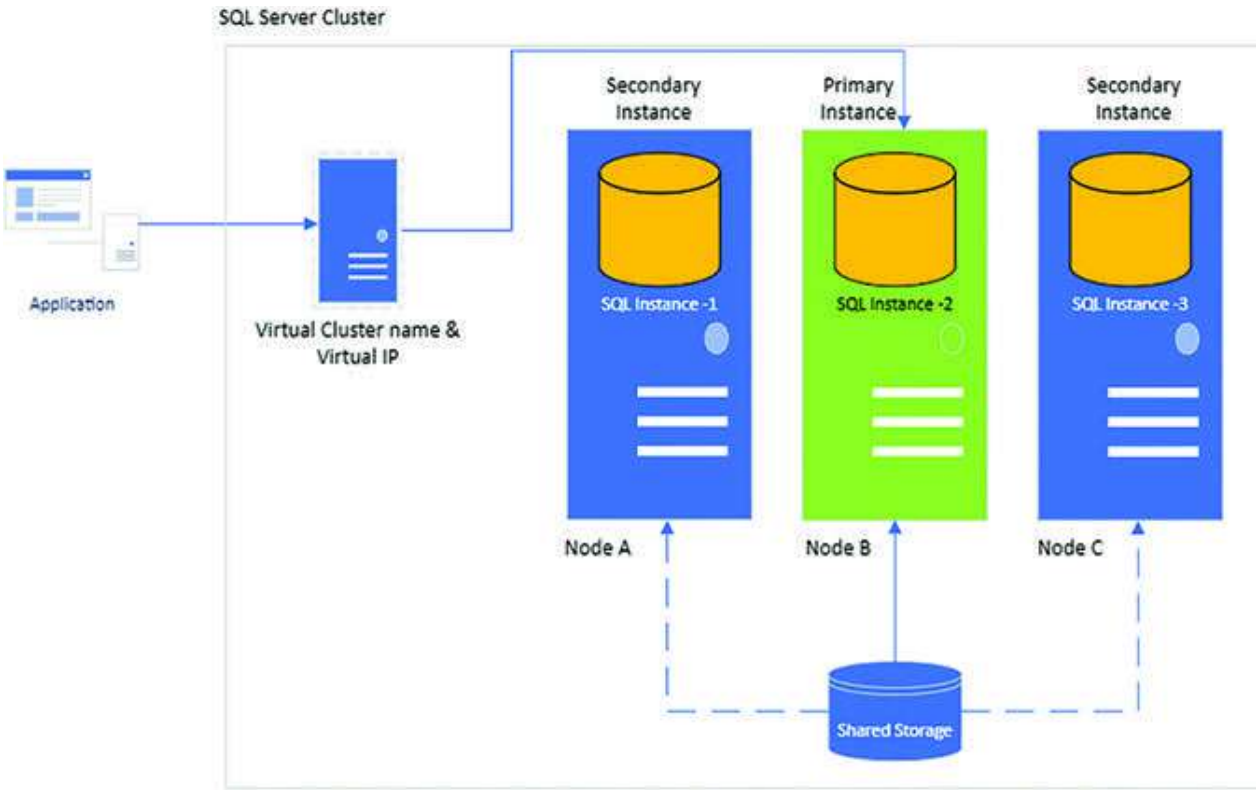


Figure 5.12: SQL Server Failover Clustered Instance after a failover, Node B is new primary

SQL Server on Linux specific

Now that we understand the cluster stack used when deploying SQL Server FCI on Windows Server Failover Cluster (WSFC), let's take a closer look at

how this stack differs for SQL Server FCI deployments on Linux environments.

There are several different cluster stacks available in the Linux ecosystem that can be used to deploy SQL Server FCI. The supported cluster stacks are:

DH2i For details on how you can get started with the FCI implementation on the DxEnterprise cluster stack, please refer to this official documentation from DH2i: HA Instances for Linux | DH2i Support Portal

HPE For instructions on how you can deploy SQL Server FCI on HPE Serviceguard as the cluster stack, refer to the official documentation from HPE Workload for Microsoft SQL Server Always on Failover Instance | HPE Serviceguard Solutions for Microsoft SQL Server for Linux User Guide

For details on the implementation of the SQL Server FCI on pacemaker, please refer to the official documentation available at Microsoft: Configure Failover Clustered Instance for SQL on Linux

Each cluster stack supports specific Linux distributions. For example, Table 5.1 shows the various distributions that are supported on each cluster stack as of writing this book:

book:
book:

Table 5.1: Supported Distributions by common cluster stack for SQL Server on Linux

Please consult the official documentation for each cluster stack to get the latest information on supported distributions.

The major differences that you should be aware of, between the SQL Server failover clustered instance (FCI) deployment on a WSFC cluster stack versus Linux cluster stack are:

For WSFC, you install SQL Server FCI as part of the installation process, and then you keep adding nodes to the existing SQL Server FCI cluster. But, for Linux, you basically configure the cluster after installing SQL Server as a standalone instance on all the required nodes.

On Linux environments, you can only have one (default) instance of SQL Server installed; there are no named instances of SQL Server. Thus, the SQL Server Failover cluster instance (FCI) is a default instance. But, on the other hand, for SQL Server on Windows, it supports up to 25 SQL Server clustered instances on every Windows Server Failover cluster.

The virtual cluster name, also called the common name in Linux, is defined in DNS and is similar to the resource created for the FCI.

Let us understand the basic cluster architecture of a Pacemaker stack. Please remember that Pacemaker is an open-source cluster stack. As described in the official documentation from ClusterLabs 1. Introduction — Clusters from Scratch the architecture basically consists of the following important parts:

This is the brain that processes and reacts to events like node joining, leaving, and resources failing.

This component provides messaging, membership, and quorum information about the cluster.

Resource Agents: These are the scripts that help manage the resource. For example, in the case of SQL Server, the resource agent is mssql-server-ha, and the resource is a SQL Server instance.

Cluster tools: These are the tools that help you in interacting, managing, and configuring the cluster. Examples include pcs and crmsh.

Figure 5.13 depicts the basic architecture stack for SQL Server FCI deployed on Pacemaker cluster stack with each of the above components.

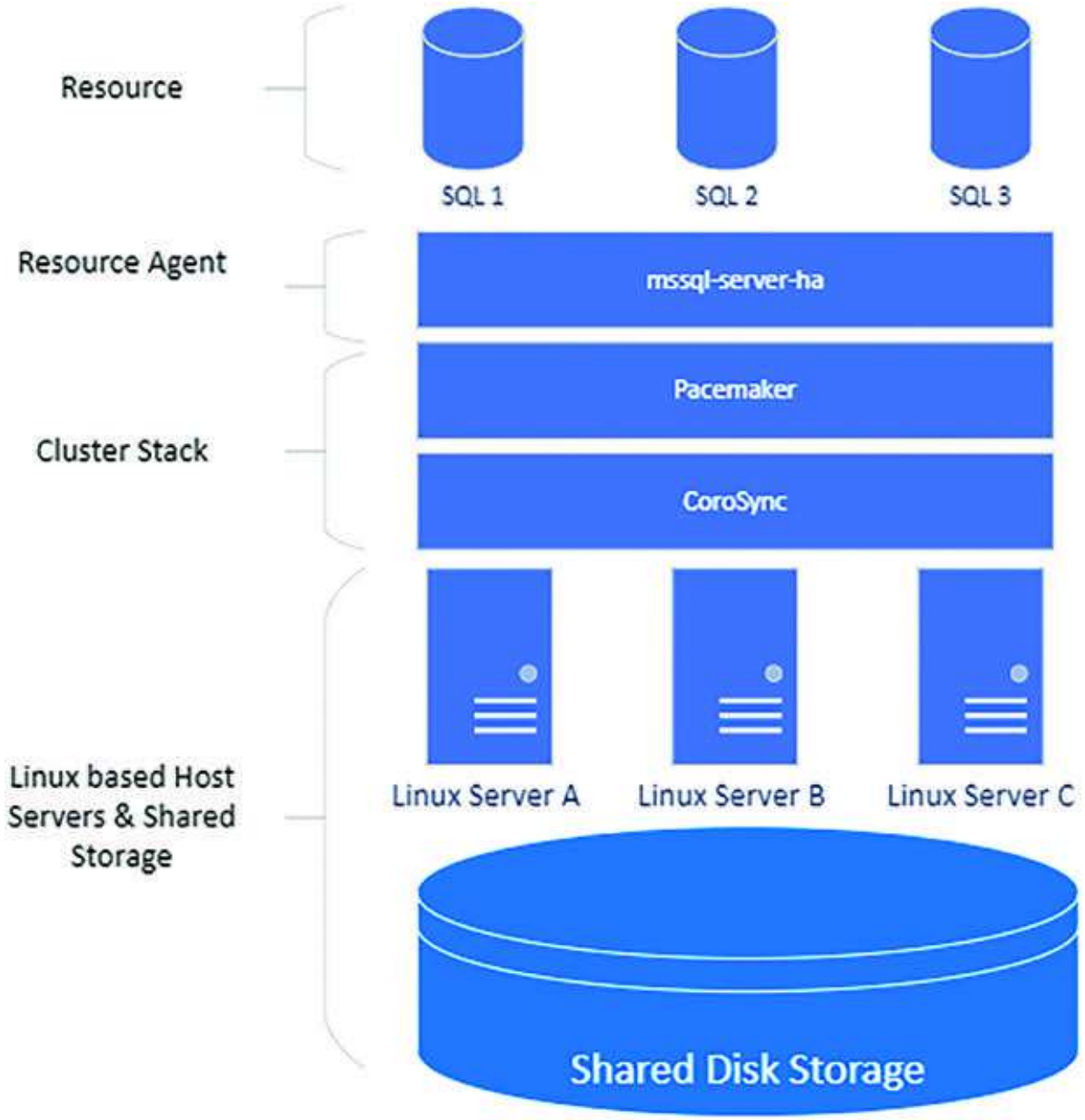


Figure 5.13: SQL Server on Linux(FCI) with pacemaker cluster using Shared storage

For SQL Server on Linux, the shared storage options include iSCSI, Network File System (NFS), and Server Message Block (SMB). If you are looking to have multi-region, multi-data center nodes for the cluster, then you need an external method for storage replication to ensure that it takes care of sharing the same storage across the multi-data center, multi-locations.

You can download the cluster stack—in this case, the packages required for Pacemaker and corosync—based on the distribution you choose. For example, for Red Hat Enterprise Linux (RHEL), you will need to enable the High Availability repo and then download the Pacemaker packages, such as the Pacemaker, pcs, and others.

Once you have the cluster stack installed and configured, you can then go ahead and install the SQL Server resource agent that is mssql-server-ha from the Microsoft repo.

That is it now; you can get started with the configuration. As documented in the official documentation from Microsoft: Configure RHEL FCI for SQL Server on Linux - SQL Server | Microsoft Learn the basic steps involved in creating a SQL Server FCI on Pacemaker are:

Install and configure SQL Server on each cluster node; this involves installing SQL Server standalone instances on each node. Also creating a login that can be used by the pacemaker to connect to each SQL Server instance.

Configure shared storage and move the database files to the shared storage to ensure that every SQL Server instance is looking at the same data.

Install and configure Pacemaker on each cluster node and install the SQL Server resource agent that is

Create the cluster resources that includes:

SQL Server Resource A name for the clustered SQL Server resource; this should be the same as the common name that was pre-created in the DNS.

Floating IP Resource A name for the virtual IP Address resource.

IP Address: The IP address that clients use to connect to the clustered instance of SQL Server. Again, the same IP Address that was added in the DNS.

File System Resource Name: A name for the filesystem resource that contains the device representing the share path and the local path that's mounted to the share as the common storage for the nodes and the file share type (NFS, iSCSI, SMB).

Finally, go ahead and create the fencing agent. A fencing agent is used to isolate a node when it is not behaving properly, such as being unresponsive, corrupted, or compromised. This is to ensure that the cluster remains consistent and reliable. There are different methods to fence a node, such as rebooting, shutting down, or disconnecting. For example, the fencing agent available in Azure for RHEL is called the: fence-agents-azure-arm.

Hopefully, now you are aware of the SQL Server FCI deployments on Windows Server Failover Cluster (WSFC) and Linux-based external clusters such as Pacemaker.

SQL Server FCIs offer server-level protection, which means that instance-level objects and settings, such as logins, jobs, Linked Servers, and more, are also preserved during a failover. The storage cost is low, as the same shared storage is used across the nodes, and the implementation is a simple HA/DR solution. However, a major drawback of SQL Server FCIs is that the that the reliance on shared storage can become a vulnerability, acting as a potential single point of failure., Unless additional storage backup solutions are employed that adds to the overall cost. Moreover, when deploying SQL Server FCIs across multiple locations or data centers, the cost of sharing the

storage using replication technologies can be high. Therefore, it is advisable to use other HA/DR features, such as Always On Availability groups with Always On failover cluster instance, to avoid the dependency on the shared storage and to create a cost-effective HA/DR solution that spans multiple sites.

Always On Availability Groups (AG)

Moving on to another feature within the Always On suite, we have Availability Groups (AGs). This technology offers high availability and/or disaster recovery at the database level. Always On availability groups were first introduced with SQL Server 2012; before this version, another option that provided database availability in SQL Server was called Database Mirroring. If you are interested in learning more about Database Mirroring, you can refer to the official documentation from Microsoft on Database Mirroring that is available here: Database Mirroring for SQL Server

Please be aware that Database Mirroring is a feature that is being phased out and will be eliminated in an upcoming version of Microsoft SQL Server.

Thus, you should not be deploying Database Mirroring anymore and instead should look to use Always On availability groups, which is a better alternative to Database Mirroring.

In an availability group, a database or a group of databases can be failed over from one SQL Server instance to another SQL Server instance. Unlike the SQL Server failover cluster instance (FCI) in availability groups (AGs), we do not have shared storage; instead, every instance of SQL Server has its own storage and set of databases that can be completely different from the other instance, except for the databases that are part of the availability groups.

If a database is part of the availability group, then it can be writable only on one SQL Server instance called the primary replica, and the other instances

will hold the copy of the database that is in sync with the primary replica depending on the node of the always on availability group configured.

Let us understand the terminology and the stack.

Availability Group This is a container that holds a single database or group of databases that fail over together to another SQL Server instance. In Figure AG1 is an example of the availability group. Please note that since DB1, DB2, and DB3 are all part of the same AG, called AG1, they can only be moved as a group; in other words, you cannot have a situation where DB1 is primary on node A, but the other two databases, DB2 and DB3, are primary on another node. Since they are part of the same AG, they failover and failback together.

Availability A database that is part of the AG and maintains read-write only on one of the SQL Server instances is called an availability database. In Figure the databases DB1, DB2, and DB3 are all part of the single availability group AG1, and are called availability databases.

Primary The availability database that allows both read and write. In Figure the node A, where all the three databases, DB1, DB2, and DB3 allow the read-write queries, is called the primary database.

Secondary Database: The availability database that only allows read queries. In Figure the databases DB1, DB2, and DB3 on nodes B and C are the secondary databases.

Replicas: Any SQL Server instance running on a host that is participating in availability groups is called as a Replica. In Figure the nodes A, B, and C are called Replicas, as each of these nodes hosts its own SQL Server instances.

Primary The SQL Server instance (replica) that hosts the primary database(s) and thus allows read and write queries on the availability databases. It also sends the transaction log records for each primary database to every secondary replica. In Figure the SQL Server instance running on Node A is the primary replica.

Secondary The SQL Server instance or instances (Replicas) that host the secondary database(s) and serve as potential failover targets. Optionally, you could also configure the secondary databases to allow read-only queries or run backup jobs to create the backups of the availability databases on secondary, thus reducing workload on the primary. In Figure the SQL Server instances on Node B and Node C are called the secondary replicas.

Availability Group A common server name, linked with an IP address, is used to route incoming connections either to the primary replica or to a read-only secondary replica. Think of this like the virtual server's name in case of failover cluster instance (FCI). In Figure AG1 listener is the availability group listener, and as you can see, it is routing the read-write queries to Node A and all the read-only queries to Node B.

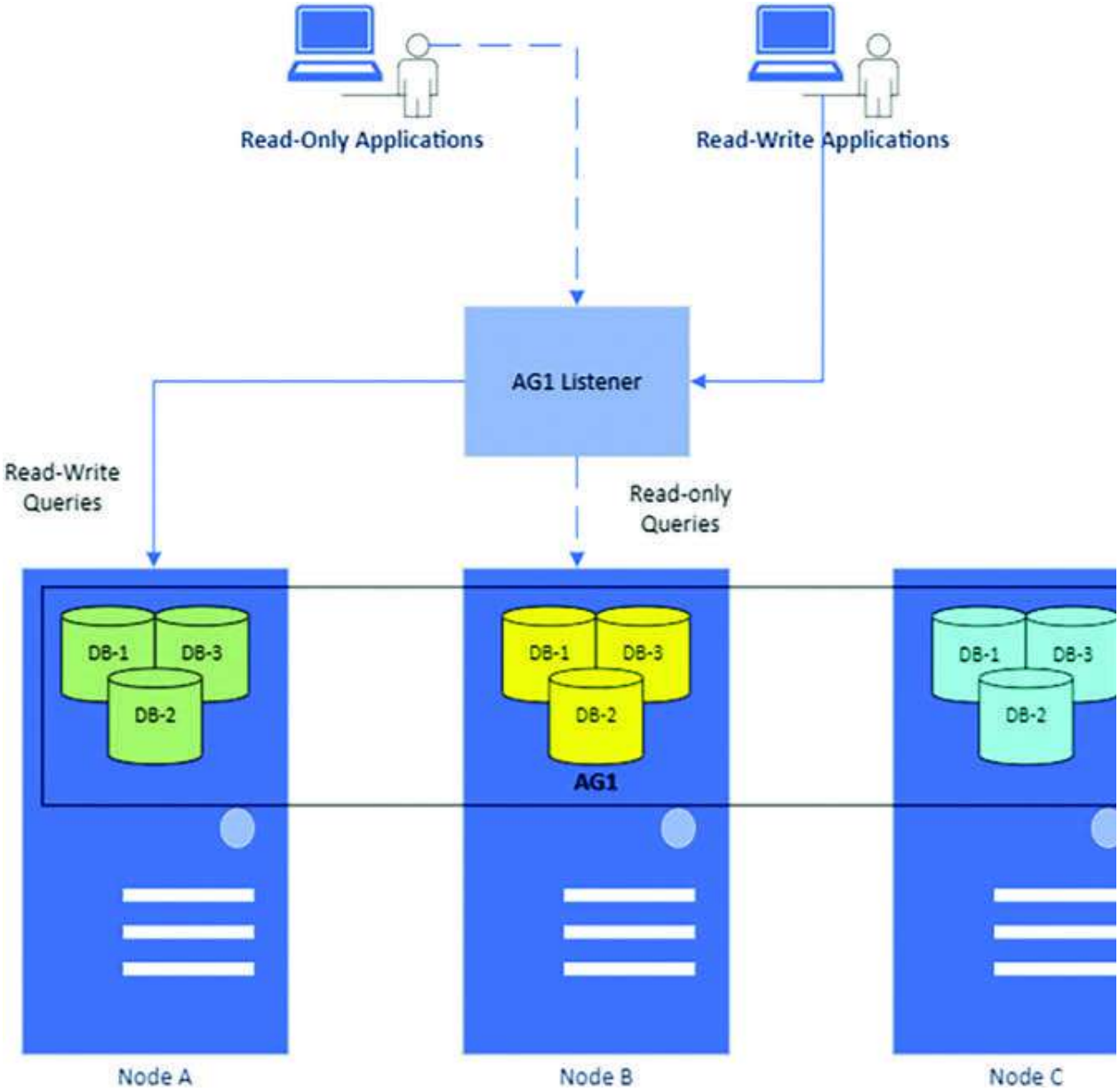


Figure 5.14: Availability Group Listener connecting to AG1 which is primary on Node A

In Figure the primary replica is Node A, which is why the availability databases DB1, DB2, and DB3 are depicted in green, indicating that these databases are online and available for read write queries. On the two secondary replicas, Node B and Node C, the availability databases are represented in yellow and blue, respectively. Node B is set to synchronous mode and Node C to Asynchronous mode.

Availability Modes:

The three availability modes that you can choose from when configuring always on availability groups are shown in Figure

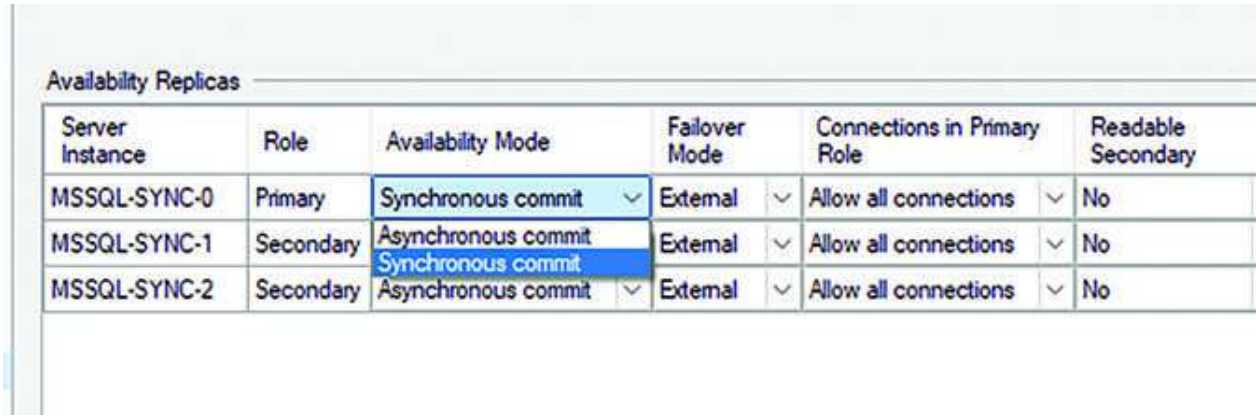


Figure 5.15: Availability modes supported by availability groups in SQL Server

Synchronous Mode: In this mode, for every transaction happening on the primary replica, the primary replica waits for a commit to be acknowledged by all the secondary synchronous commit replicas, meaning it waits for the hardening of the log (log flush, a process in SQL Server where log writes are pushed from log buffer to disk) to first happen on all the secondary replicas that are configured in synchronous mode, and then finally the primary replica commits and acknowledges to the application.

This is done to ensure that the availability database is always updated and in sync with the primary. This comes at an extra cost of transaction latency. This mode is normally used when you want to configure the database in high availability mode, such that if the primary goes down, immediately the secondary that is synchronized can be promoted to the new primary with minimum downtime for the application.

The database state on the secondary replicas, when it is in sync with the primary, is set to SYNCHRONIZED state. As shown in Figure the availability mode for the secondary replica MSSQL-SYNC-1 is set to synchronous, and the database state is synchronized.

Asynchronous mode: In this case, the primary replica commits transactions without waiting for the acknowledgement from the replicas that are configured as asynchronous. Thus, in this case, the primary replica does not wait for the logs to be hardened on the asynchronous secondary replicas. Thus, in this case, there is no transaction latency as the primary is not waiting on the asynchronous secondary replicas, but the asynchronous secondary replicas may lag the primary database, making some data loss possible.

The database state on the asynchronous secondary replicas is always in SYNCHRONIZING state. In Figure the MSSQL-SYNC-2 is an asynchronous secondary replica, and the database is in SYNCHRONIZING state.

You would normally use the asynchronous secondary replica for disaster recovery situations and not for high availability.

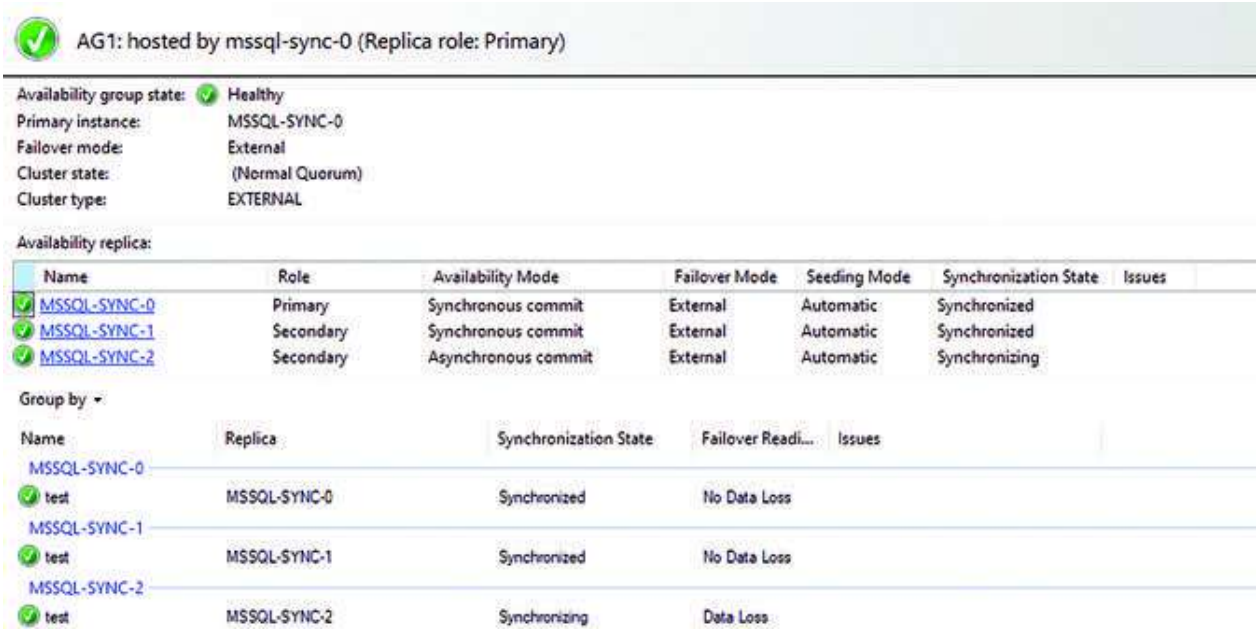


Figure 5.16: Availability group dashboard

Configuration Only This mode is available only when you deploy availability groups that do not use Windows Server Failover Cluster (WSFC) as the cluster stack and you are running SQL Server 2017 CU 1 and above. A replica configured as configuration only does not host the availability databases but only contains the metadata for the availability group that is synchronously committed. This metadata is required for assistance in forming quorum and supporting automatic failovers. Any edition of SQL Server can be used to create configuration-only replica, including the SQL Server Express edition. We will discuss more about this in the upcoming section of the chapter when we talk about automatic failovers.

Understand failover in availability groups:

By now, you know that in a failover cluster instance (FCI), a failover of the instance basically means that the SQL Server instance goes down or offline on one node and then comes online on the other node, but in the context of availability groups, failover is the process where one node becomes the

primary replica for the availability databases, thus allowing read-write queries, and the other node becomes the secondary replica and optionally allowing read-only queries.

During the failover process, the new primary replica brings the availability databases online as the primary database, and the former primary replica, when it is available, joins the availability group as a secondary replica. The former primary databases become secondary databases, and data synchronization resumes.

Please note that the failovers can be triggered for issues at the database level, such as a database becoming suspect due to loss of a data file, deletion of a database, corruption of a transaction log, or the database not being online. This can be enabled by checking the option database-level health detection when configuring the availability group, as shown in Figure You need to tread this option carefully. If you select it and your availability group contains multiple databases, the automatic failover will be triggered if even one database is offline. This can result in downtime for all the other databases in the availability group.

If you do not select the database-level health detection, then database issues do not trigger the failover, but only SQL Server instance level issues trigger the failovers.

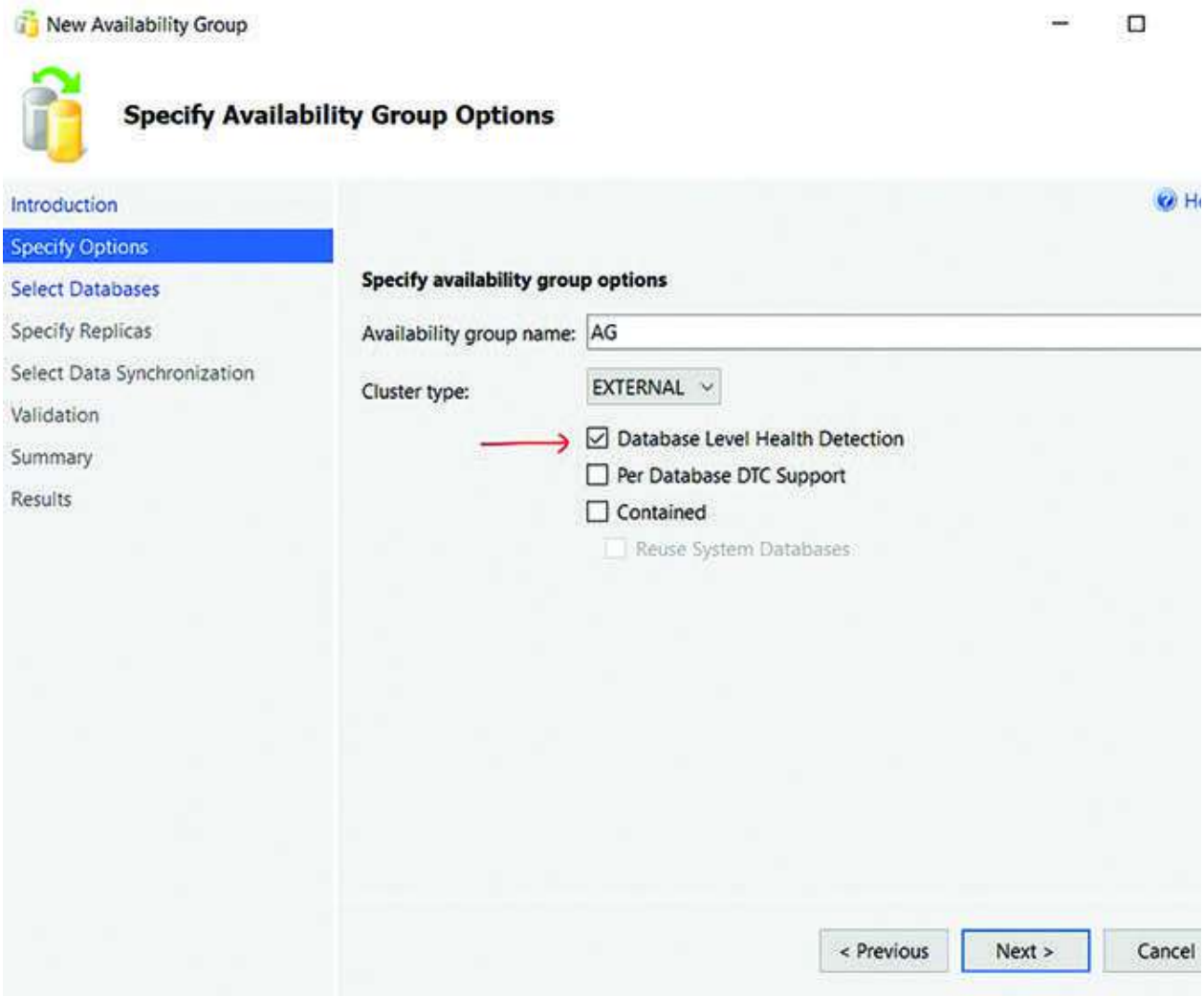


Figure 5.17: Opting for Database Level heath detection

In the context of Availability Groups (AGs), there are various failover types that can be utilized when Synchronous commit mode is active. This mode primarily supports two forms of failover, which are:

Planned manual failover (without data loss)

As the term implies, a manual failover is initiated by executing the failover command, which allows a synchronized secondary replica to assume the primary role, ensuring data protection. Data protection is guaranteed by confirming that both the primary replica and the target secondary replica

operate under synchronous-commit mode and that the secondary replica (the prospective new primary replica) is already synchronized. Once the original primary replica is back online, it rejoins the availability group as a secondary replica and synchronizes with the new primary.

Automatic failover (without data loss)

In this scenario, the failover occurs automatically, prompting a synchronized secondary replica to take over the primary role, with data protection assured. Once the original primary replica is back online, it switches to a secondary role. For an automatic failover to happen, two specific conditions need to be fulfilled:

The primary replica and the target secondary replica must be operating under synchronous-commit mode with the failover mode set to automatic.

The secondary replica should already be synchronized, possess a WSFC quorum, and comply with the conditions outlined by the availability group's flexible failover policy. For an in-depth explanation of the flexible failover policy, refer to the official Microsoft documentation: The flexible failover policy for an availability group is determined by its specified failure-condition level and the health-check timeout threshold. When it is detected that an availability group has surpassed its failure condition level or the health-check timeout threshold, the resource DLL of the availability group communicates this to the Windows Server Failover Clustering (WSFC) cluster. Subsequently, the WSFC cluster triggers an automatic failover to the secondary replica.

You may ask, what is the health-check timeout threshold and failure-condition level? As explained in the aforementioned Microsoft official article and following mentioned for easy reference:

Health-check timeout threshold:

“WSFC resource DLL of the availability group performs a health check of the primary replica by calling the sp_server_diagnostics stored procedure on the instance of SQL Server that hosts the primary replica. sp_server_diagnostics returns results at an interval that equals ۱/۳ of the health-check timeout threshold for the availability group. The default health-check timeout threshold is ۳۰ seconds, which causes sp_server_diagnostics to return at a 10-second interval. If sp_server_diagnostics is slow or is not returning information, the resource DLL will wait for the full interval of the health-check timeout threshold before determining that the primary replica is unresponsive. If the primary replica is unresponsive, an automatic failover is initiated, if currently supported.”

Failure-condition level:

Whether the diagnostic data and health information returned by sp_server_diagnostics warrant an automatic failover depends on the failure-condition level of the availability group. The failure-condition level specifies what failure conditions trigger an automatic failover. There are five failure-condition levels, which range from the least restrictive (level one) to the most restrictive (level five). A given level encompasses the less restrictive levels. Thus, the strictest level, five, includes the four less restrictive conditions, and so forth.

The following Figure 5.18 describes the failure condition that corresponds to each level.

Level	Failure-condition	Transact-SQL Value	PowerShell Value
One	On server down. Specifies that an automatic failover is initiated when one the following occurs: The SQL Server service is down. The lease of the availability group for connecting to the WSFC cluster expires because no ACK is received from the server instance. For more information, see How It Works: SQL Server Always On Lease Timeout. This is the least restrictive level.	1	OnServerDown
Two	On server unresponsive. Specifies that an automatic failover is initiated when one of the following occurs: The instance of SQL Server does not connect to cluster, and the user-specified health check timeout threshold of the availability group is exceeded. The availability replica is in failed state.	2	OnServerUnresponsive
Three	On critical server error. Specifies that an automatic failover is initiated on critical SQL Server internal errors, such as orphaned spinlocks, serious write-access violations, or too many memory dumps generated in a short period of time. This is the default level.	3	OnCriticalServerError
Four	On moderate server error. Specifies that an automatic failover is initiated on moderate SQL Server internal errors, such as a persistent out-of-memory condition in the SQL Server internal resource pool.	4	OnModerateServerError
Five	On any qualified failure conditions. Specifies that an automatic failover is initiated on any qualified failure conditions, including: Detection of Scheduler deadlock. Detection of an unsolvable deadlock. This is the most restrictive level.	5	OnAnyQualifiedFailureConditi

Figure 5.18: Different failover-conditions levels in SQL Server

Manual Failover (with possible data loss):

This is the only failover option possible under the asynchronous-commit mode, meaning that this failover is possible when the target secondary replica is not synchronized in a synchronizing state with the primary replica. This is also known or called forced failover mode. It is basically a disaster recovery option. Whenever this failover occurs, there is always a possibility of data loss as the new primary is not in complete sync with the old primary.

Availability Groups Cluster types and Improvements starting SQL Server 2017 for Linux support:

Before SQL Server 2017, availability groups required a Windows Server Failover Cluster (WSFC). Starting with SQL Server 2017, availability groups can also be used on Linux distributions. This is possible because of new features and enhancements to availability groups, including:

Cluster Types: There are three options that you can set the cluster type to when configuring SQL Server availability groups, and they are:

WSFC (can only be used when you are deploying AGs on Windows clusters).

External (to be used when you are deploying AGs on a Linux-based cluster stack).

None (to be used when you intend to deploy AGs for read-scale configuration).

As shown in Figure 5.19 which is referenced from the official Microsoft documentation on availability groups overview for SQL Server on Linux: https://learn.microsoft.com/en-us/sql/linux/sql-server-linux-business-continuity-dr?view=sql-server-ver16

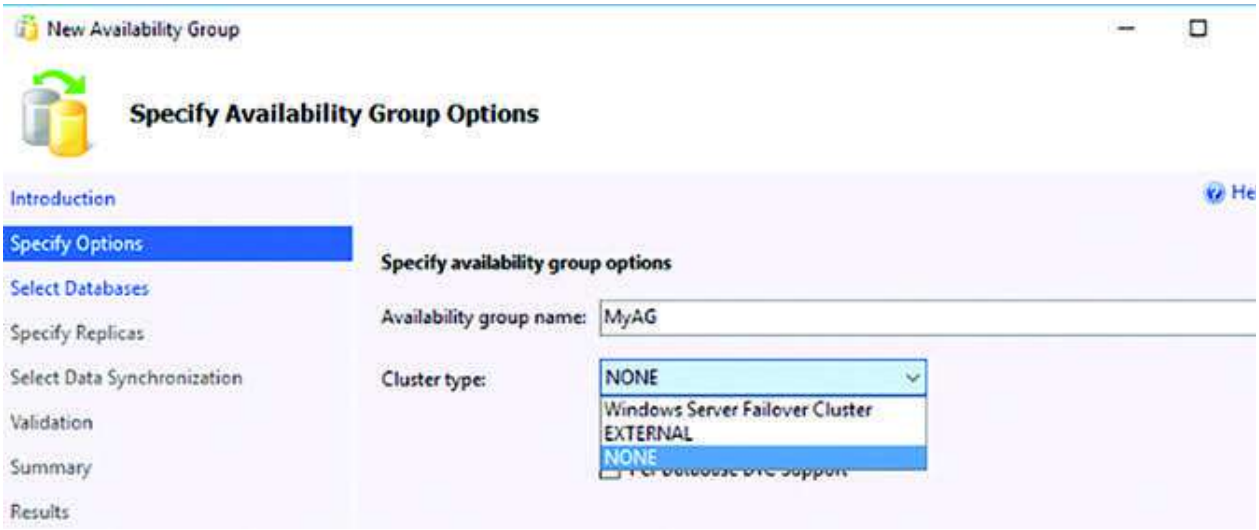


Figure 5.19: Different cluster types supported by SQL Server

To support a non-Windows cluster stack, a new cluster type option was added called External. External clusters support Linux-based cluster stacks such as HPE Serviceguard, DH2i DxEnterprise, and Pacemaker for availability groups. Also, a new cluster type called None was introduced to support setting up availability groups without any cluster.

Required_secondaries_to_commit: This setting in availability groups conveys the number of secondary replicas that must be in synchronized state with the primary. It controls the availability of the primary database if the right number of secondary replicas are offline or online.

In SQL Server 2017 and onward versions, the behavior of synchronous replicas can be managed using This feature operates as follows:

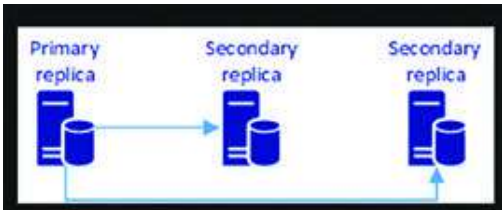
There are three potential values: 0, 1, and 2.

These values determine the number of secondary replicas that must be synchronized, affecting the potential for data loss, the availability of the Availability Group (AG), and the failover process.

For Windows Server Failover Clusters (WSFCs) and a cluster type of 'None', the default is set to 0 but can be manually adjusted to 1 or 2. In contrast, for a cluster type of 'External', the cluster mechanism typically sets this value by default, although it can be manually overridden. For instance, with three synchronous replicas, the default value would be set to 1. The official documentation of AG deployment pattern from Microsoft showcases the different configurations and how these settings affect those specific configurations. Please refer to the following link:

https://learn.microsoft.com/en-us/sql/linux/sql-server-linux-availability-group-ha?view=sql-server-ver16

The following explanation provides a quick reference to understand how the primary replica's availability functions in each of these configurations. Three Synchronous replicas:



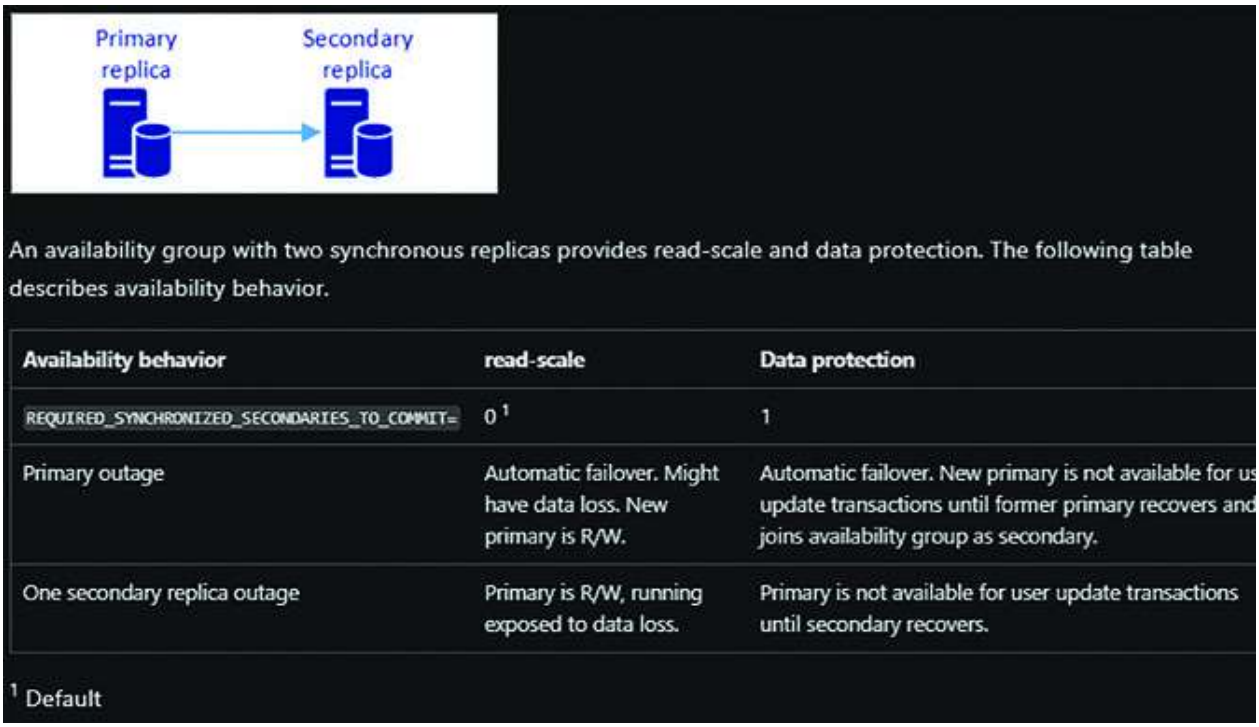
An availability group with three synchronous replicas can provide read-scale, high availability, and data protection. The following table describes availability behavior.

Availability behavior	read-scale	High availability & data protection	Data protection
`REQUIRED_SYNCHRONIZED_SECONDARIES_TO_COMMIT=`	0	1 [1]	2
Primary outage	Automatic failover. Might have data loss. New primary is R/W.	Automatic failover. New primary is R/W.	Automatic failover. New primary is not available for user update transactions until former primary recovers and joins availability group as secondary.
One secondary replica outage	Primary is R/W.	Primary is R/W.	Primary is not available for user update transactions until failed secondary recovers and joins availability group.

[1] Default

Figure 5.20: Three Synchronous Replica configuration for SQL Server on Linux

Two Synchronous replicas:



An availability group with two synchronous replicas provides read-scale and data protection. The following table describes availability behavior.

Availability behavior	read-scale	Data protection
REQUIRED_SYNCHRONIZED_SECONDARIES_TO_COMMIT=	0 [1]	1
Primary outage	Automatic failover. Might have data loss. New primary is R/W.	Automatic failover. New primary is not available for us update transactions until former primary recovers and joins availability group as secondary.
One secondary replica outage	Primary is R/W, running exposed to data loss.	Primary is not available for user update transactions until secondary recovers.

[1] Default

Figure 5.21: Two Synchronous Replica configuration for SQL Server on Linux

Two Synchronous Replicas and a Configuration only

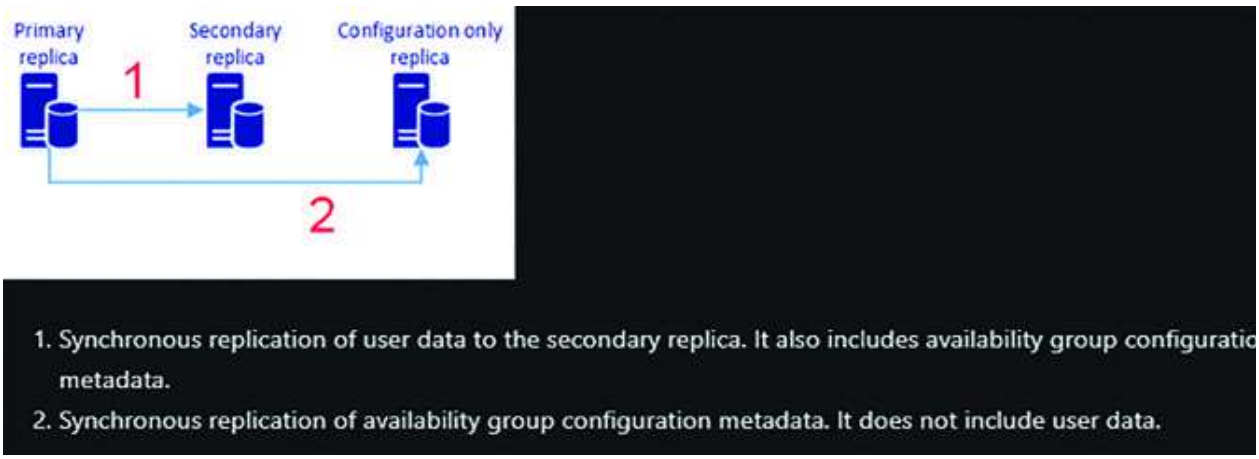


Figure 5.22: Two synchronous and a configuration-only replica configuration for SQL Server on Linux

Configuration-only Replica and need for quorum for SQL Server on Linux deployments:

Let us take this opportunity to talk a bit more about configuration only replica. Configuration only replica is the mode possible only for availability group deployments in SQL Server on Linux. For Windows-based always on availability group deployment, you do not need a configuration only replica because, in the case of the WSFC, you can have an automatic failover possible with only two replicas and a quorum that is formed by the cluster. For an Availability Group (AG) on Linux, the arbitration process is managed within SQL Server, where all the relevant metadata is stored. This is the role of the configuration-only replica, which is crucial in this context.

As previously stated, the configuration-only replica retains the AG configuration within the master database, similar to other replicas within the AG setup. However, the key difference is that the configuration-only replica does not involve any user databases in the AG.

For automatic failover to occur in Availability Groups (AGs) deployed on SQL Server on Linux, one of the following conditions must be met:

There should be three synchronous replicas, which is an option exclusive to SQL Server Enterprise, or

There should be two replicas, one primary and one secondary, along with a configuration-only replica.

This ensures that the necessary mechanisms are in place for a seamless transition during an automatic failover event.

Read Scale Availability Groups: When you set the cluster type to None and then configure the availability group on Windows or Linux, this AG does not provide high availability but only provides read scale routing and options for manual failover with or without data loss. You can use this form of AG deployment when you want to redirect the read-only queries from primary replicas to secondary replicas. Thus, helping you reduce the workload on the primary replica.

Finally, to summarize, there are a few key differences to be aware of when configuring and managing availability groups on Windows and Linux cluster stacks.

When you are looking to use availability groups with a Linux-based cluster stack, you need to use the cluster type as external.

Due to the differences in the underlying cluster management on Linux and Windows Servers, all failovers, whether manual or automatic, for Availability Groups (AGs) on Linux are conducted through the cluster management tool. On the other hand, for Windows-based clusters, manual failovers must be executed via SQL Server, while the cluster stack manages automatic failovers. For automatic failover to work in SQL Server on Linux deployments of availability groups, you need to have at least three synchronous replicas or two synchronous replicas and a configuration-only replica.

Finally, the common name used by each listener is defined in DNS on Linux and not in the cluster, unlike in Windows, where it is defined inside the cluster.

Configure Always On availability groups on Windows:

AGs on SQL Server deployed on Windows have been available since SQL Server 2012; for detailed steps, please refer to the official Microsoft document which talks on how you can configure Always On availability group for SQL Server on

Configure Always On availability group on Linux:

The focus for this section is how you can get started with availability groups for SQL Server on Linux. As of writing this book, AGs are supported for the following Linux cluster stacks; there is not a single cluster stack that is native, meaning unlike Windows cluster, there is no first party cluster stack that is available for SQL Server. Thus, with all these cluster stacks, SQL Server and availability group components are supported by Microsoft, and all the cluster stack, management, and quorum components are supported by the company that provides the cluster stack.

DH2i DxEnterprise

HPE Serviceguard

Pacemaker

Each of these Linux cluster stacks supports AGs on specific operating systems and SQL Server-specific deployments. Thus, based on your requirements, you can choose the cluster stack that suits your needs the best. For example, if you are looking to deploy SQL Server across containers, Linux, and Windows operating systems in your environment, then you might

choose the DH2i DxEnterprise, as that gives you the flexibility to deploy AG on any operating system and any SQL Server Deployments.

Similarly, if you already are using HPE Serviceguard as the high availability solution for your other database-specific workloads, such as SAP Hana, then you might want to continue with HPE Serviceguard as the HA solution for SQL Server as well.

Figure 5.23 summarizes the different Linux cluster stacks, supported SQL Server deployment environments, and supported operating systems.

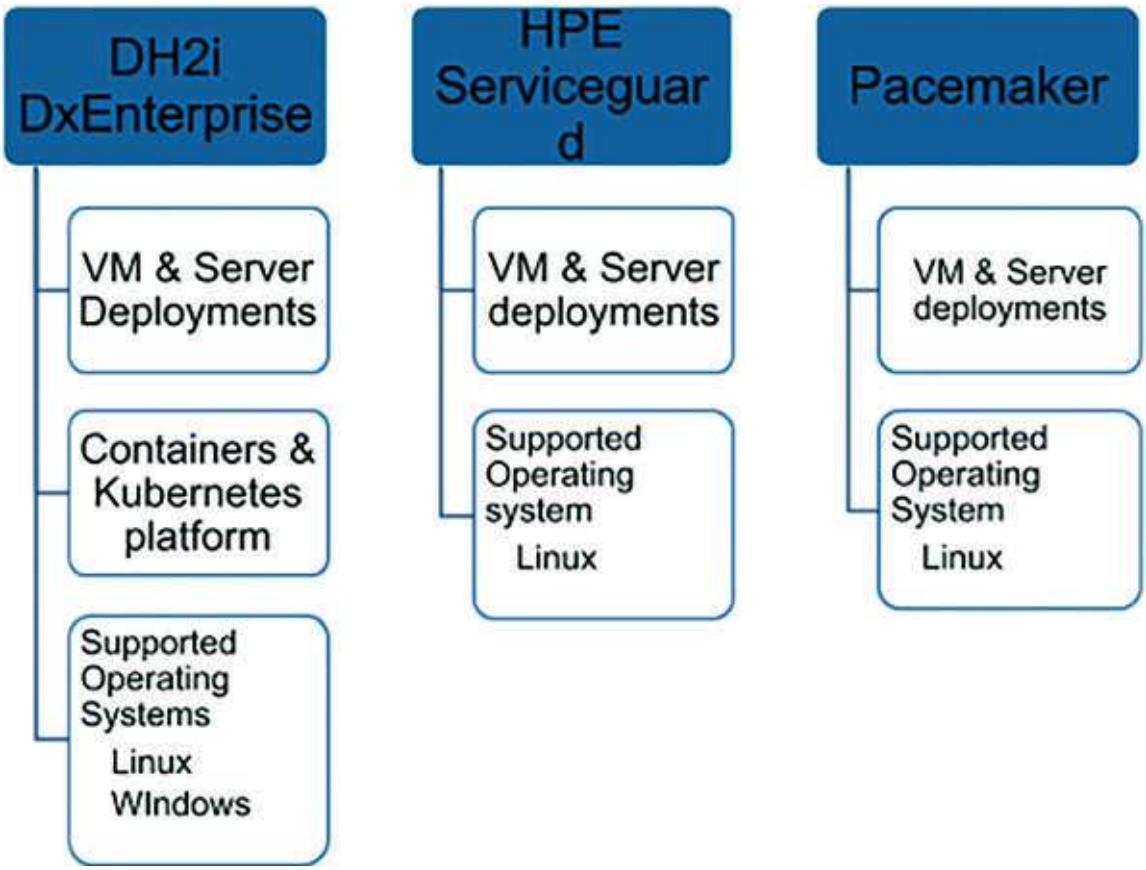


Figure 5.23: Common cluster stacks supported by SQL Server on Linux across various deployments

To deploy SQL Server availability groups on Pacemakers or HPE Serviceguard (SGLX) with three replicas, the following general steps should be taken, assuming that three virtual machines (VMs) are already available. Install and configure SQL Server on Linux on all the three VMs.

Now configure and create the availability group on these VMs; that involves enabling the Always on Availability group feature, creating the certificates and endpoints on all three VMs, creating the availability group, and then joining the nodes to the availability group. For SQL Server on Linux, the always on endpoints are authenticated via a certificate; as of writing this book, Windows authentication-based endpoints are not supported yet.

Install and configure the cluster, which could be Pacemaker or HPE Serviceguard based on your choice.

Finally, create the AG specific resources and constraints inside the cluster stack.

For Pacemaker cluster, to make all this easy and automated, there is an Ansible script that does all the aforementioned four steps without requiring any manual intervention. For details please follow the steps in the Github chapter 5 repo for this book: SQL-Server-Unveiled---Path-to-Data-Excellence/chapter 5 at main · ava-orange-education/SQL-Server-Unveiled---Path-to-Data-Excellence (github.com) https://github.com/ava-orange-education/SQL-Server-Unveiled---Path-to-Data-Excellence/blob/main/chapter%205

Once the deployment completes, you can now check the Pacemaker cluster status and AG status from the SSMS, as shown in Figure

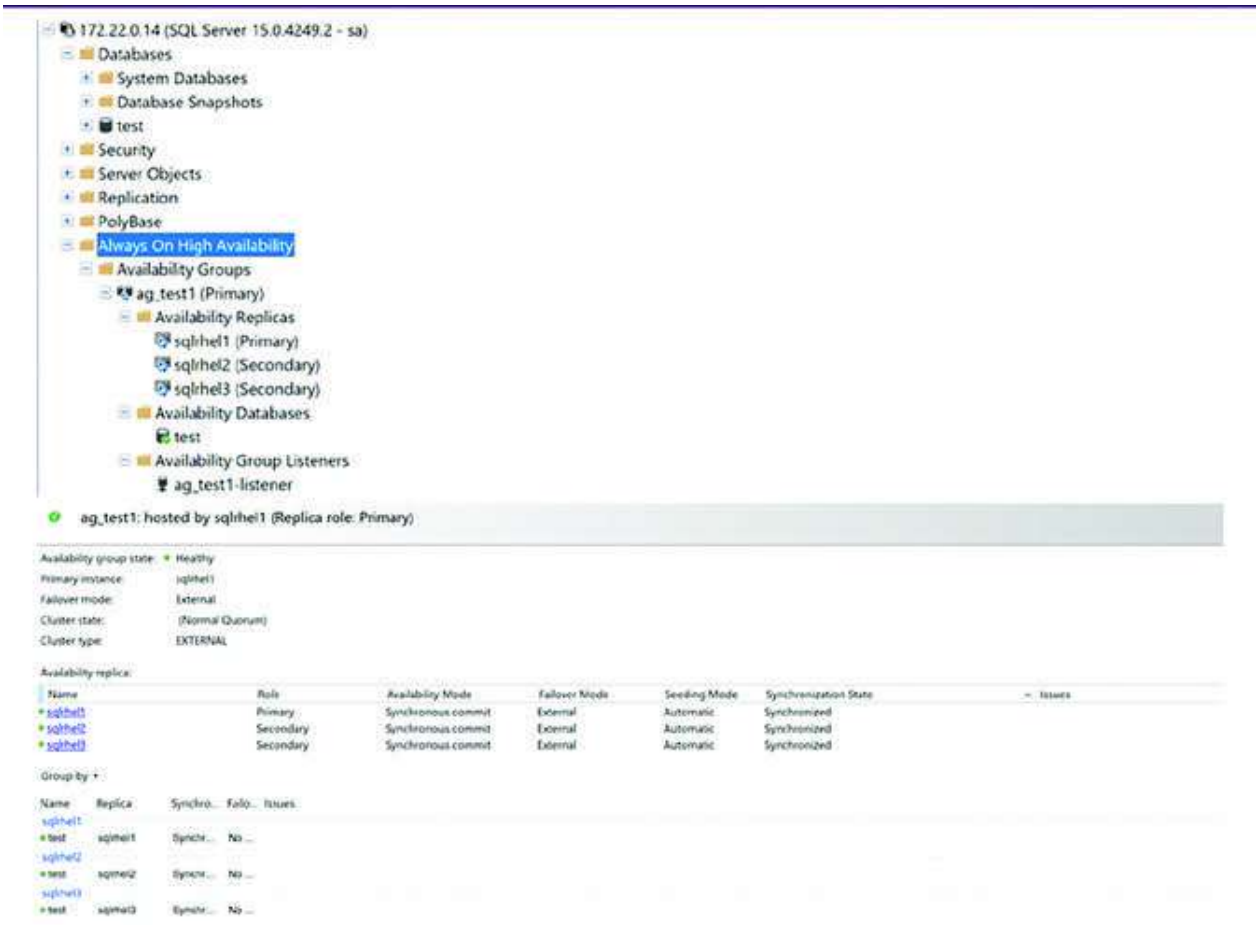


Figure 5.24: Always On Dashboard in SSMS showing the ag_test1 configured.

Similarly, you can also deploy HPE Serviceguard-based availability group for SQL Server on Linux using the Ansible script shared here: serviceguard/ansible-sglx at Stable-v1.2 · HewlettPackard/serviceguard (github.com)

Thus, using Ansible, you can make your deployments automated and scalable.

Availability groups for SQL Server Containers deployed on Kubernetes.

You can also deploy availability groups as a high availability feature for SQL Server containers deployed on Kubernetes, which is supported for production workloads.

When you are running SQL Server containers on Kubernetes, as shown in Figure the pod-level high availability is provided by the replica set in the Kubernetes. For databases running inside the pod running the SQL Server container, you can use availability groups for high availability, which provides you with the RPO and RTO targets that you expect from databases.

When you deploy SQL Server containers in Kubernetes environments, normally one pod runs one SQL Server instance via the container image. If you are looking to identify the best practices to run SQL Server containers on Kubernetes, you refer to the official Microsoft documentation available here at: Deploy SQL Server containers on Kubernetes with StatefulSets - SQL Server | Microsoft Learn

We will discuss this in detail in the upcoming chapters of this book, where we specifically look at the best practices for SQL Server.

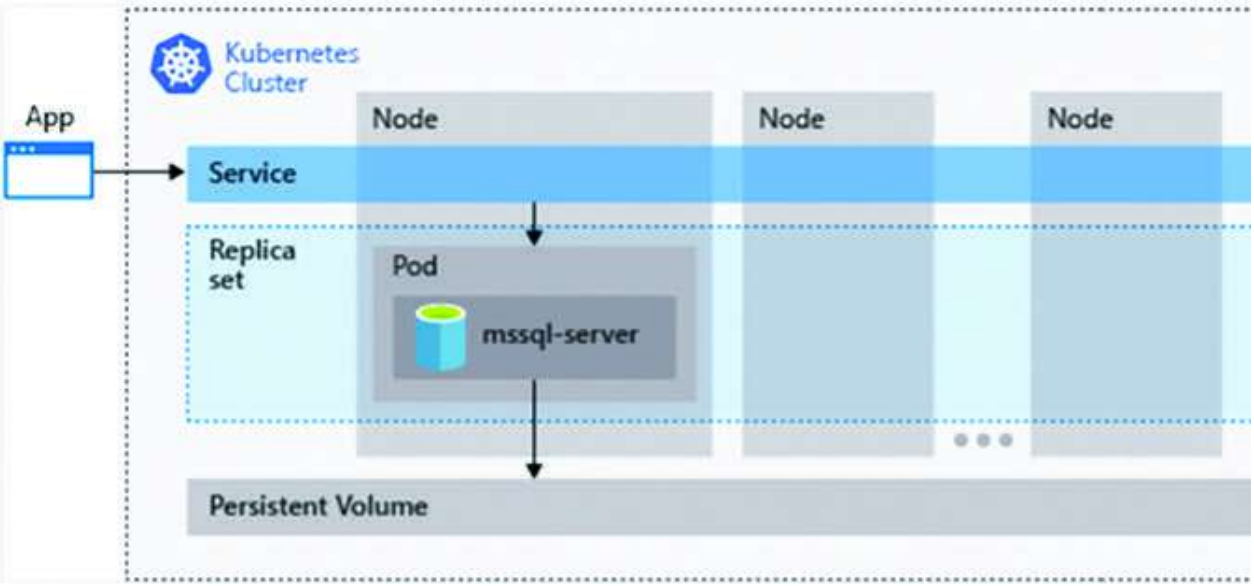


Figure 5.25: SQL Server containers deployed on Kubernetes Cluster

As of writing this book, the only cluster stack that works for SQL Server containers across Kubernetes clusters is DH2i DxEnterprise. You can setup the availability group for SQL Server containers on Kubernetes by following the steps listed in the official documentation from Microsoft: Deploy availability groups with DH2i DxEnterprise on Kubernetes - SQL Server | Microsoft Learn As you may notice, there are two ways to deploy, and they are:

Sidecar container This is a simple option, where inside the script you deploy SQL Server as the main container and for the high availability cluster stack, you deploy the DH2i-specific packages as side containers that provide the HA stack. It is recommended to use this as you can update the SQL Server on DH2i packages independently without having to recreate the container images.

Custom container In this mode, you end up creating a custom container image that you deploy, which in turn deploys the SQL Server instance and the DH2i packages. You will need to upload this custom container image to a central container registry if you intend to share the images. Thus, this method required a lot of maintenance and configuration.

Once you successfully configure the availability group by following the steps as described in the official Microsoft documentation, you can see the AG status using the AG dashboard available in SQL Server Management Studio (SSMS), as shown in Figure Please note that all three instances are three different SQL Server pods, each pod running one SQL Server instance inside the same Kubernetes cluster.

In this case, the Kubernetes cluster being run is a cluster on Azure Kubernetes Service (AKS), and once you glance at the pods, you will see three pods running, as shown in Figure

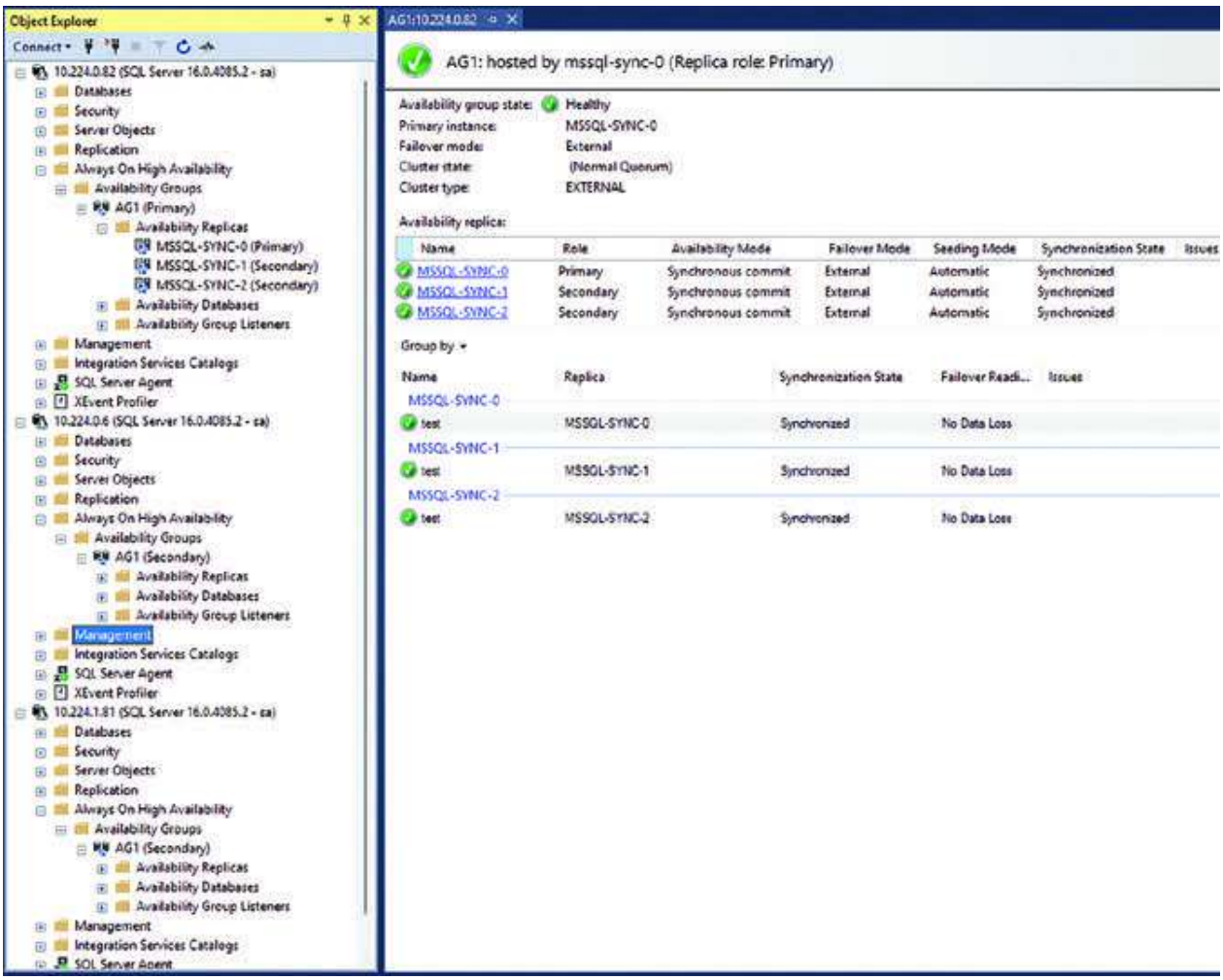


Figure 5.26: Always On availability group dashboard in SSMS, showing SQL Pods part of the AG replica

You can also run the kubectl commands to view the pods, their status, and different services and ports associated with them using the command: kubectl get all, as shown in Figure

```
C:\Users\amitkh>kubectl get all
NAME                  READY   STATUS    RESTARTS   AGE
pod/mssql-sync-0      2/2     Running   0          6d4h
pod/mssql-sync-1      2/2     Running   0          6d4h
pod/mssql-sync-2      2/2     Running   0          6d4h

NAME                          TYPE           CLUSTER-IP      EXTERNAL-IP     PORT(S)                                        AGE
service/kubernetes            ClusterIP      10.0.0.1        <none>          443/TCP                                        89d
service/mssql-sync-0          ClusterIP      None            <none>          7979/TCP,7980/TCP,7981/UDP,5022/TCP,1433/TCP   6d4
service/mssql-sync-0-lb       LoadBalancer   10.0.153.137    52.172.38.103   7979:30940/TCP,1433:31487/TCP                  6d4
service/mssql-sync-1          ClusterIP      None            <none>          7979/TCP,7980/TCP,7981/UDP,5022/TCP,1433/TCP   6d4
service/mssql-sync-1-lb       LoadBalancer   10.0.52.233     13.71.116.159   7979:30163/TCP,1433:31329/TCP                  6d4
service/mssql-sync-2          ClusterIP      None            <none>          7979/TCP,7980/TCP,7981/UDP,5022/TCP,1433/TCP   6d4
service/mssql-sync-2-lb       LoadBalancer   10.0.236.90     13.71.126.222   7979:30324/TCP,1433:31086/TCP                  6d4
```

Figure 5.27: Kubectl output showing SQL pods and services

Cross platform availability group deployment for high availability:

Because SQL Server is supported across multiple operating systems and deployment patterns, such as SQL Server on Windows, SQL Server on Linux, or SQL Server containers deployed across the Kubernetes platform. You also have the ability to configure the AGs across replicas that are running on different platforms, as shown in Figure

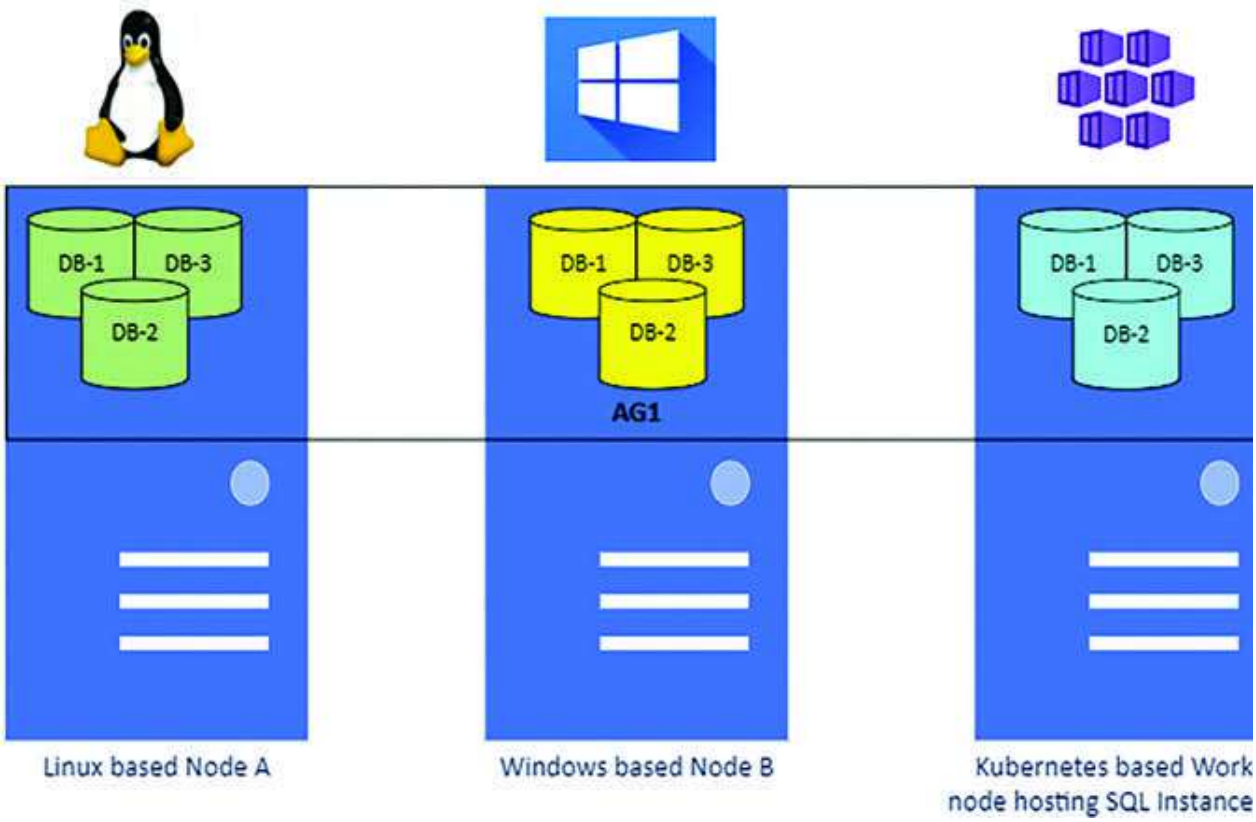


Figure 5.28: Cross platform availability group setup

You can deploy Always On availability group for SQL Server across cross platform in two ways:

Read Scale mode deployment of It is mostly used for moving the read only workloads away from the primary and distributing them to read only secondaries. Or you could also use this mode for planning and executing SQL Server migration from one SQL Server deployment to another. To enable this mode, you need to set the cluster type to None and then go ahead with the configuration of the availability group across different SQL Server deployments. For details on the steps required to configure this, please refer to the official Microsoft documentation as detailed here: Deploy availability groups with DH2i DxEnterprise on Kubernetes - SQL Server | Microsoft Learn

High availability and Disaster Recovery mode: This is the general use case where you can have the replicas of SQL Server running three different deployments, such as one replica running SQL Server on Windows, another replica running SQL Server on Linux, and the third replica running SQL Server on Containers, and have an availability group that supports automatic failover of the availability databases across any of the replicas. This today is possible using the DH2i DxEnterprise cluster stack, and this is a fully supported solution stack, as documented in this official Microsoft documentation: Configure SQL Server Always On availability group on Windows and Linux - SQL Server | Microsoft Learn To configure the AG in high availability mode across cross platform, please refer to the official documentation rom DH2i: Availability Groups with Mixed OS | DH2i Support Portal

Distributed Availability Groups:

Hopefully, by now you are already aware of how to configure always on availability groups for SQL Server on Windows, Linux, or containers deployed on the Kubernetes platform. You also understand the concepts of failover, cluster types, and different types of replicas that you can configure when deploying AGs. All these configurations focused on either high availability, read scalability across replicas, or both. But we did not talk about how you can achieve Disaster Recovery with availability groups yet, and this is where Distributed availability groups (DAGs) come into play.

Distributed availability groups (DAGs) are basically used to enable the three main scenarios, and they are:

Disaster Recovery across multi-site configurations.

Migration to new hardware or configuration, which might include changing the underlying operating system or using new hardware.

Increasing the number of readable replicas beyond eight in a single availability group spanning multiple availability groups.

Now, before we look at each of these scenarios, it is important to understand how the Distributed Availability Groups (DAGs) basically work. A DAG is a special type of availability group that spans two separate availability groups. The AGs that participate in DAGs can be geographically located at different locations and can also run different operating systems.

Another important aspect of DAGs is that it does not configure anything for which a cluster stack is required. In other words, automatic failover is not

supported and is not possible. All the metadata required to maintain a DAG is stored within SQL Server.

To understand this further, let us look at Figure as documented in the official Microsoft documentation here at: What is a distributed availability group - SQL Server Always On | Microsoft Learn

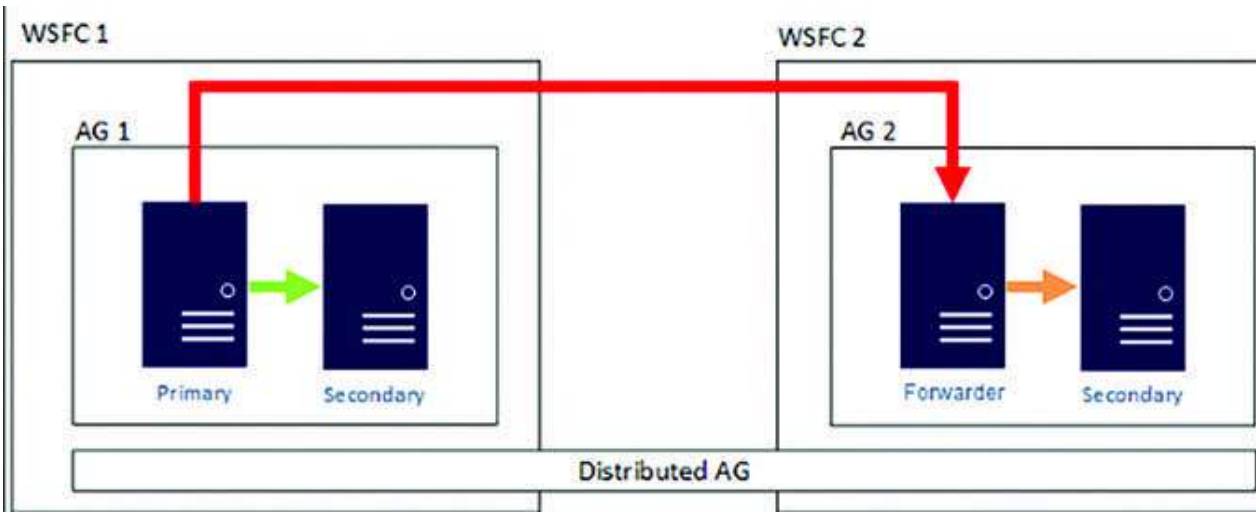


Figure 5.29: Distributed Availability Groups

As represented in Figure you basically have two Windows Server Failover cluster (WSFC) that are probably running across data centers in different cities. So, this can be a multi geographically located data center. Here are the key points to understand about the DAGs:

You have two availability groups, AG1 and AG2, running on their own cluster stack that could be a Windows-based WSFC stack or could be a Linux cluster stack such as Pacemaker.

The availability database in both these AGs (AG1 and AG2) is the same database or databases.

The AG1 is the primary AG, within this DAG. For every transaction happening on the primary replica of AG1, it sends these transactions to the secondary replica of AG1 and sends them to the primary replica of AG2. This primary replica of AG2 is also called the forwarder. You cannot run read-write queries on the forwarder; the forwarder can only be used for read-only queries as it always tries to be in sync with the primary of AG1.

The forwarder then takes those transactions and sends them to its own secondaries.

Based on the synchronous or asynchronous mode that you set for the secondaries, the primary from AG1 accordingly sends the transactions to the secondary of its own AG and to the forwarder of the other AG that is part of the DAG.

Manual failover is possible in DAGs where you can make the AG2 primary accept the read-write queries if needed, and then the primary of AG1 becomes the forwarder and is given the responsibility to keep its secondaries updated.

Hence, the DAGs are also called the availability group of availability groups. For more information on how you can configure this for different scenarios, it is recommended that you refer to the official documentation from Microsoft here at: What is a distributed availability group - SQL Server Always On | Microsoft Learn

New features for AG in SQL Server 2022.

With the release of SQL Server 2022, one of the most asked features was made available with the availability groups, and that was the ability to not

only replicate user databases but also metadata objects such as users, logins, SQL agent jobs, and other instancelevel context with the database. This feature is introduced under the name Contained Availability Group (CAG).

Remember, this feature of contained availability groups is completely different from contained databases that were introduced in SQL Server 2012. A contained database has much of its metadata, settings, and configuration within itself and not inside the master database or at the instance level. Basically, with contained databases, you can avoid orphaned logins, as the login details also move with the database. Also, the user authentication can be performed by the database and does not depend on the instance. To know more about contained databases, please refer to this official Microsoft documentation: Contained Databases - SQL Server | Microsoft Learn

The contained availability group basically supports:

Managing metadata objects (users, logins, permissions, SQL Agent jobs, and so on) at the availability group level in addition to instance level.

Specialized contained system databases within the availability group.

Normally, the metadata, such as user, login, and SQL Agent jobs, all of this information is stored in the system databases, such as the master and msdb. An application that uses a user database to store user data would also need these metadata to ensure it is able to connect and work with the database. It is basically called the application context. Every application has its application context, and when a user database failover occurs, this application context has to be managed by the users manually. This is where the contained availability group comes into the picture and includes the relevant portions of the master

and msdb databases inside the AG. In other words, you can create users, logins, permissions, and SQL Agent jobs at the availability group level, and they will automatically be consistent across replicas in the availability group, as well as consistent across databases within that contained availability group. Thus, no more manual intervention is required to sync the application execution context.

You can create the contained availability group using the SSMS UI "new availability group wizard," as shown in Figure with the contained option checked.

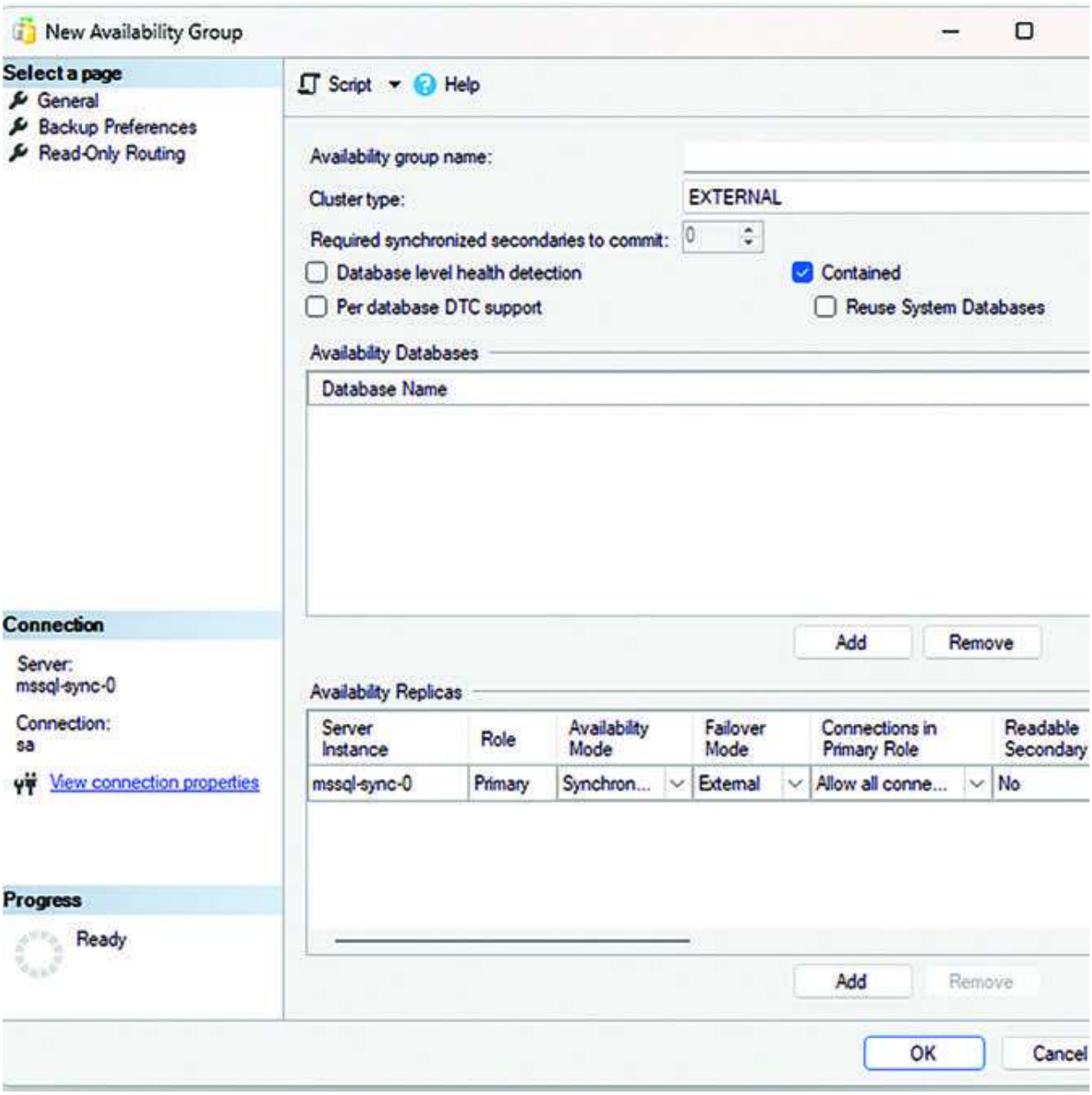


Figure 5.30: Setting up contained availability group in SSMS

Following is the T-SQL script to create the contained availability group for an external cluster type used with a Linux-based cluster stack. Remember, before running this command, you will need to create the certificates and AG endpoints on all the replicas.

```
CREATE AVAILABILITY GROUP CAGTest

WITH (CONTAINED, CLUSTER_TYPE = EXTERNAL)
FOR DATABASE cagtest REPLICA ON
N’mssql-sync-0’
WITH (
ENDPOINT_URL = N’tcp://10.224.0.75:5022’,
AVAILABILITY_MODE = SYNCHRONOUS_COMMIT,
FAILOVER_MODE = EXTERNAL,
SEEDING_MODE = AUTOMATIC
),
N’mssql-sync-1’
WITH (
ENDPOINT_URL = N’tcp://10.224.0.29:5022’,
AVAILABILITY_MODE = SYNCHRONOUS_COMMIT,
FAILOVER_MODE = EXTERNAL,
SEEDING_MODE = AUTOMATIC
),
N’mssql-sync-2’
WITH(
ENDPOINT_URL = N’tcp://10.224.1.62:5022’,
AVAILABILITY_MODE = SYNCHRONOUS_COMMIT,
FAILOVER_MODE = EXTERNAL,
SEEDING_MODE = AUTOMATIC
```

);

You can join the other replicas normally, like you would for any normal AG. These T-SQL commands need to be run on all the replicas that you intend to join the contained AGs.

ALTER AVAILABILITY GROUP CAGtest JOIN WITH (CLUSTER_TYPE = EXTERNAL);

ALTER AVAILABILITY GROUP CAGtest GRANT CREATE ANY DATABASE;

Once you successfully create a contained AG, the AG dashboard should look as shown in Figure AG1, the standard Availability Group (AG) in the environment, just houses the availability database “test.” In addition, three availability databases have been assembled into a confined Availability Group (CAG) called CAGTest. These databases are a user database called “cagtest” and two system databases called “CAGTest_master” and “CAGTest_msdb.”

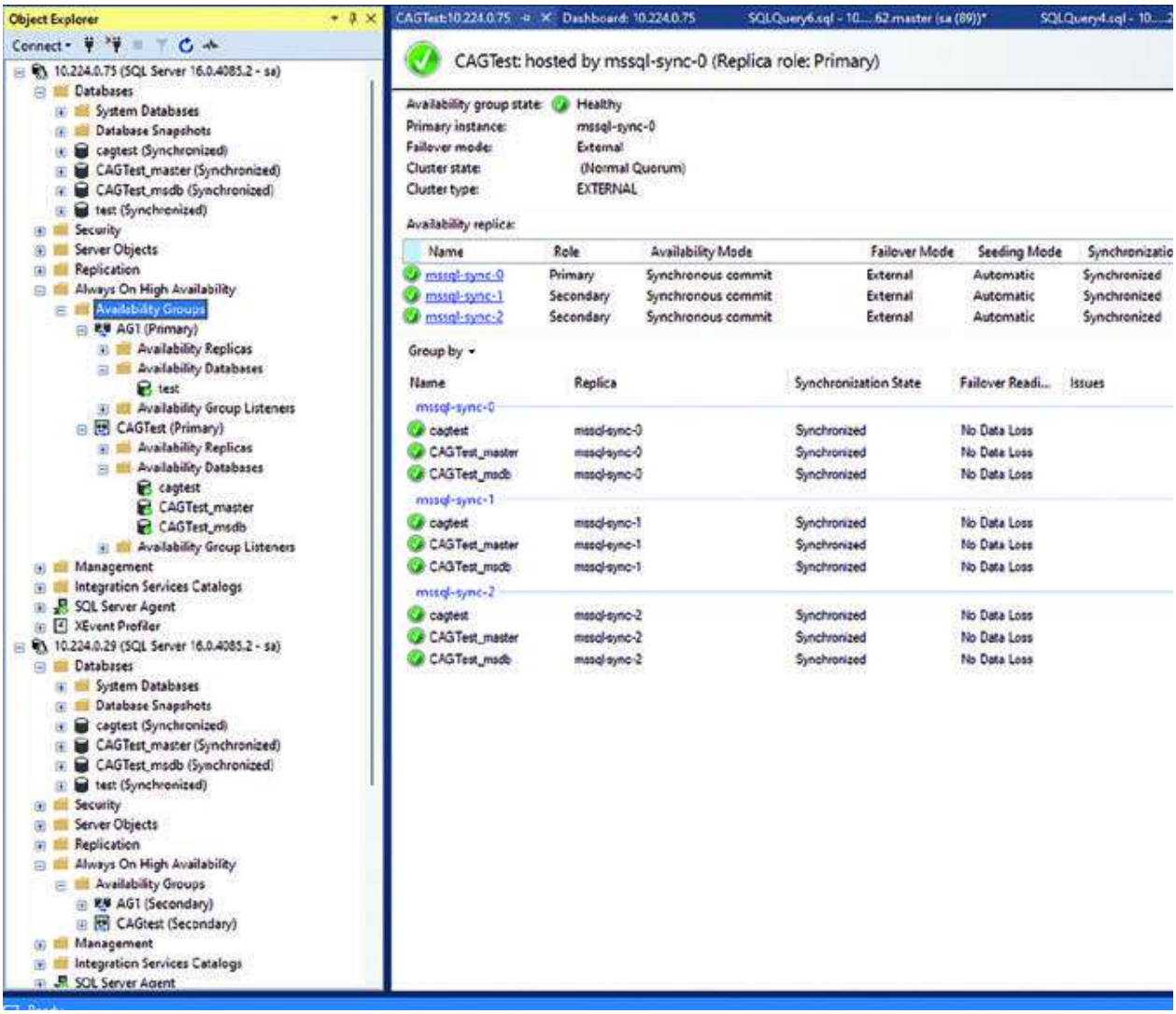


Figure 5.31: Always On availability group dashboard in SSMS

There are a few important aspects that you should know about contained availability groups.

Each contained availability group has its own master and msdb system databases that are named as contained availability group name_master and contained availability group As shown in Figure 5.31 the contained availability group name is CAGTest, and the master and msdb databases part of the CAG are CAGTest_master and CAGTest_msdb.

The only way to access the environment of the contained availability group is to connect to the contained availability group listener or to connect to a database that is in the contained availability group. When you connect to the contained availability group, then the CAGTest_master becomes the master and CAGTest_msdb becomes the msdb database, as shown in Figure Using the USE CAGTest_master command and the SELECT DB_NAME() function while connected to a confined availability group database reveals that the database name displays as master.

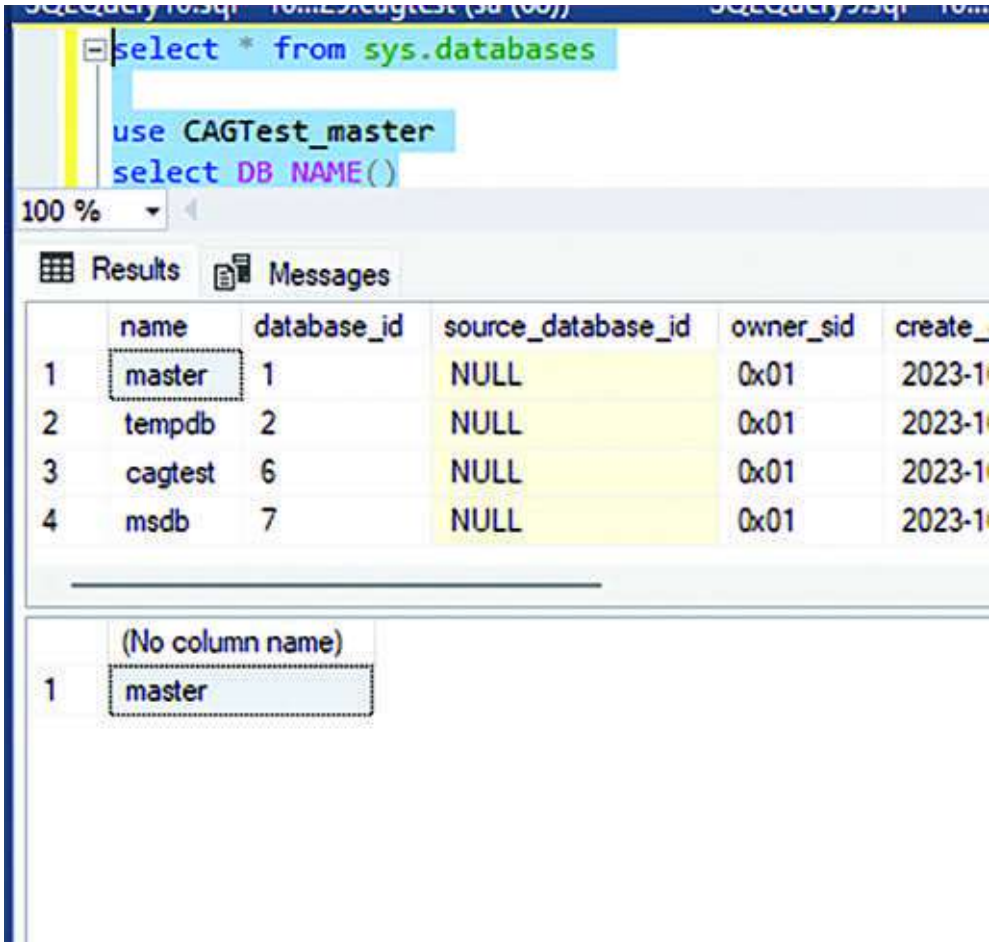


Figure 5.32: Query to show CAGTest_master as the master database with Contained AGs

By default, when you create the contained availability group, not all users and logins present on the primary replica replicate to the contained availability

group master database; only logins that are part of the sysadmin get added.

When you create the logins and users under the context of the contained availability groups, only then those logins and users get created on other replicas, as shown in Figure 5.33 and Figure Figure 5.33 represents the data from the primary replica where we created the login called

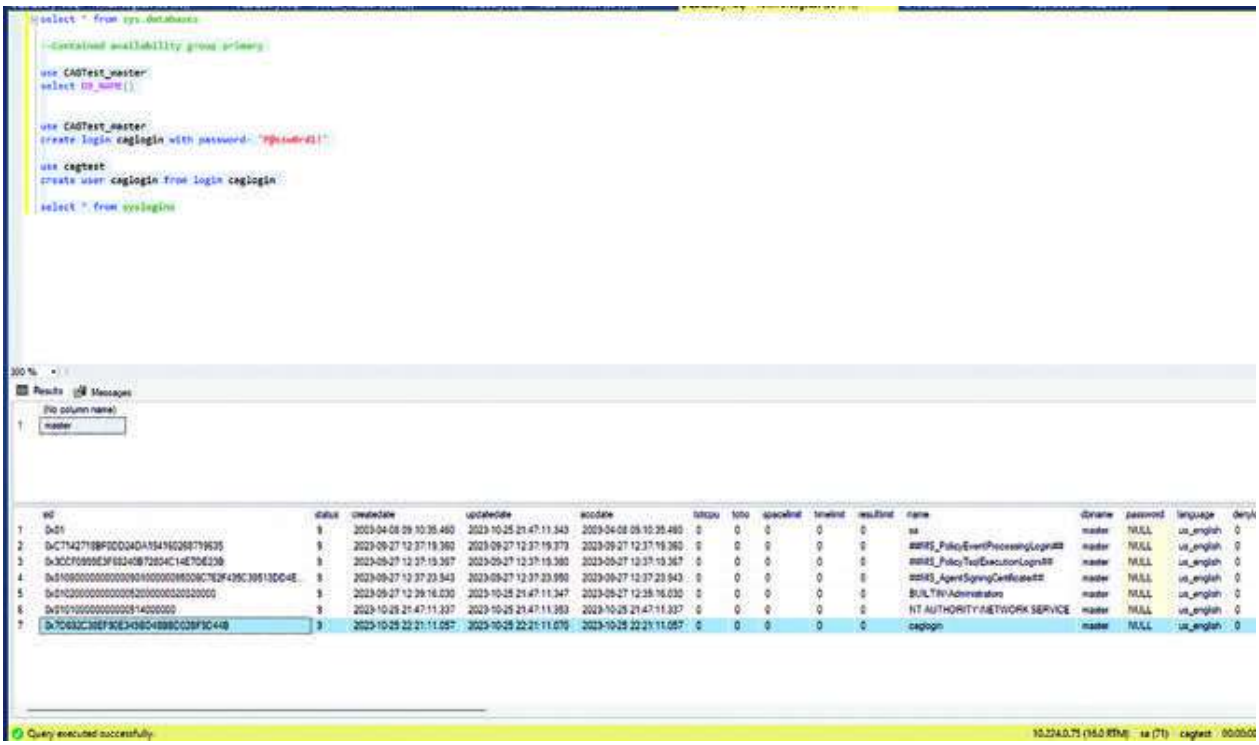


Figure 5.33: Data from the primary replica

Figure 5.34 represents the data from the secondary replica where you can see the login caglogin created.

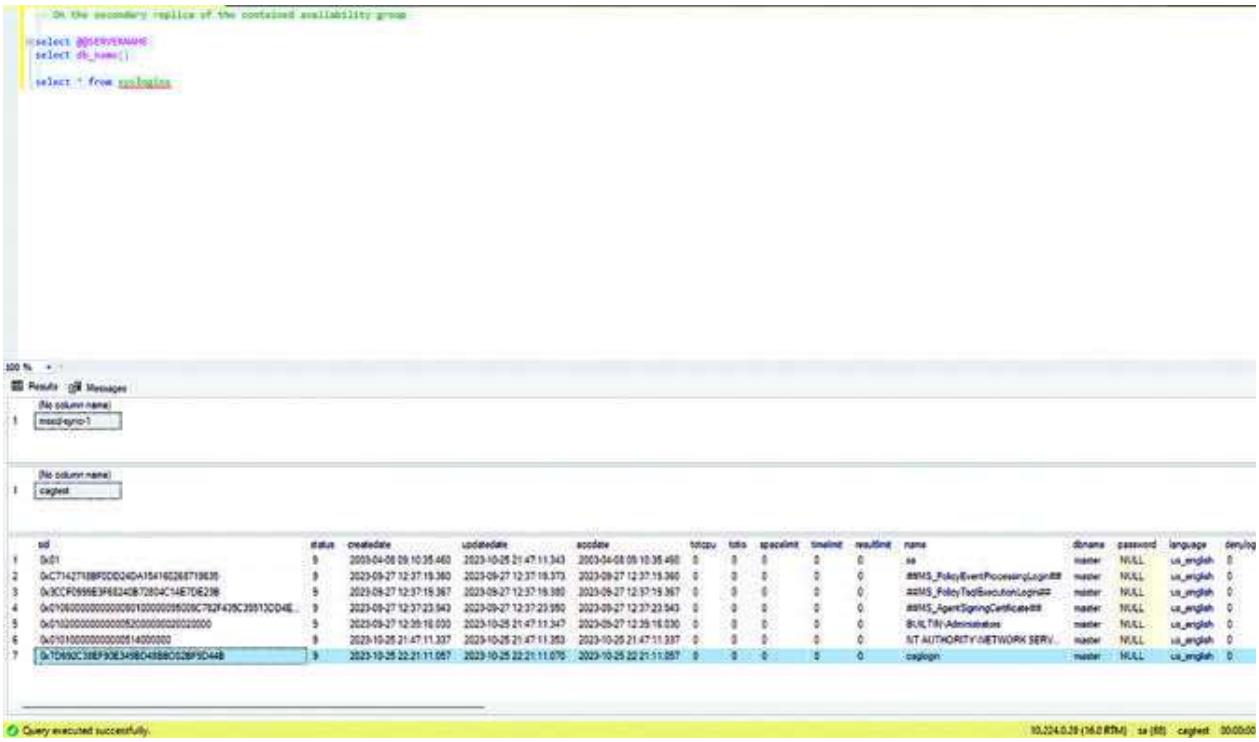


Figure 5.34: Data from the secondary replica where the new login is created

For more information, please refer to the official Microsoft documentation: What is a contained availability group? - SQL Server Always On | Microsoft Learn

This also describes some of the features that do not work with contained availability groups.

Business Continuity Strategies for Azure SQL

In all the previous sections, we have learned about the basics of most of the commonly used high availability and disaster recovery features. In this section, we look at how all the offerings within the Azure SQL family are made available to end users and customers.

Azure SQL DB and Azure SQL Managed Instance (MI)

As you are now aware, both Azure SQL DB and Azure SQL Managed Instance (MI) are the PaaS offerings of the SQL Server database engine and thus manage functions such as upgrading, patching, backups, and monitoring without much user involvement. On similar lines, any disruptive events that occur in the cloud environment can be handled by Azure SQL DB and Azure SQL MI such that your applications continue to run with minimum downtime.

But, still, for these PaaS services, you also need to plan your business continuity for situations that cannot be handled by these services on their own. The following list contains a few examples of such situations that require user intervention to create a business continuity strategy. This is not an exhaustive list, but just a few items mentioned to help you understand why you need to develop a business continuity strategy even when you are running PaaS services such as Azure SQL DB and SQL MI in the cloud.

User errors are induced by human error. Like accidentally deleting a table or updating all the rows of a table.

Malicious attack that ends up deleting, dropping, or updating tables.

Natural calamities that could temporarily take down the entire data center in that geographic region.

Upgrade or maintenance errors that cause temporary failures.

Now, to prepare for such scenarios, both Azure SQL DB and Azure SQL MI provide the following common features that, as a user, you can selectively enable based on your requirements.

Built-in automated backups and point in time restores

Restore a deleted database

Long-term backup retention

Active Geo-replication for Azure SQL DB or Managed Instance Link for Azure SQL MI

Auto failover groups

Let us look at each of these features briefly so you have the required information to build your business continuity strategy based on your business requirements.

Built-in Automated Backups and Point in Time Restores

Backup and restore is an essential part of any business continuity and disaster recovery strategy, as these fundamental aspects help protect your data from corruption or accidental deletes.

By default, Azure SQL DB and Azure SQL MI both create full backups every week, Differential backups every 12 or 24 hours, and Transaction log backups every 10 minutes. The frequency of the log backups depends on the compute size and the amount of database activity. You can also enable long term backup retention to keep the database backup beyond the 1-35 days provided by automatic backups. You can also enable backup storage redundancy locally, Zone-redundant, or Geo-redundant storage for extra precautions. To know more about how you can configure the backups and how you can restore based on your RPO and RTO requirements, go through this official Microsoft documentation: Automatic, geo-redundant backups - Azure SQL Database | Microsoft Learn

Restoring a Deleted Database

Azure SQL DB and Azure SQL MI also provide you the ability to restore from the automated backups on the same logical server or a different logical server across geographies by enabling Geo-Restore.

Geo-Restore basically uses geo-replicated backups as a source and thus gives you the ability to restore the database on any logical server in any Azure region. Kindly note that a time lag is inherent when a backup is initiated and subsequently geo-replicated to an Azure blob in another region. Consequently, the restored database may lag by up to one hour compared to the original database.

You should use Geo-restores as a disaster recovery solution. The RPO is of up to 1 hour and the estimated RTO of up to 12 hours. Also, the capacity in the target region is not guaranteed because there could be an increase in demand when there are data center failures, triggering Geo-restores to other regions. Hence, you should use this solution as a disaster recovery for non-business-critical applications and if you have small databases.

For business-critical applications, please use auto-failover groups, as this feature offers reduced RPO and RTO targets and the capacity is always guaranteed. For further information, please read: Restore a database from a backup - Azure SQL Database | Microsoft Learn

Active Geo-Replication for Azure SQL DB

This is the feature that lets you create a continuously synchronized, readable secondary database for a primary database. Think of this equivalent to the availability groups that you would configure for your on-premises deployments of SQL Server. But here you can just enable Active Geo-Replication just through a few clicks using the Azure portal. To view the steps, please refer to the official Microsoft documentation which has this explained in detail: Tutorial: Geo-replication & failover in portal - Azure SQL Database | Microsoft Learn

You can also enable this programmatically using Powershell or Azure CLI, and the steps are available in the same article referenced.

Using the Active Geo-Replication, you can configure automatic asynchronous replication, Readable geo-secondary replicas, and multiple readable geo-secondaries, all very similar to the availability group deployments. So in addition to the disaster recovery solution, you can use this feature also for database migration to migrate from one server to another with minimum downtime and also for application upgrades to create an extra secondary as a failback copy during application upgrades.

For more information on the configuration of this feature, refer to this documentation from Microsoft: Active geo-replication - Azure SQL Database | Microsoft Learn

[Auto-Failover Groups for Azure SQL DB and Azure SQL Managed Instance (MI)]

An auto-failover group in Azure SQL Database can include one or more databases, typically used by the same application. Think of it as an availability group with more than one availability database. When you enable auto failover group with automatic failover policy, an outage that impacts one or other database in the group will result in an automatic geo-failover.

The Auto-failover group must be configured on the primary server and will connect it to the secondary server in a different Azure region. Here is a diagram from official Microsoft documentation: Auto-failover groups overview & best practices - Azure SQL Database | Microsoft Learn

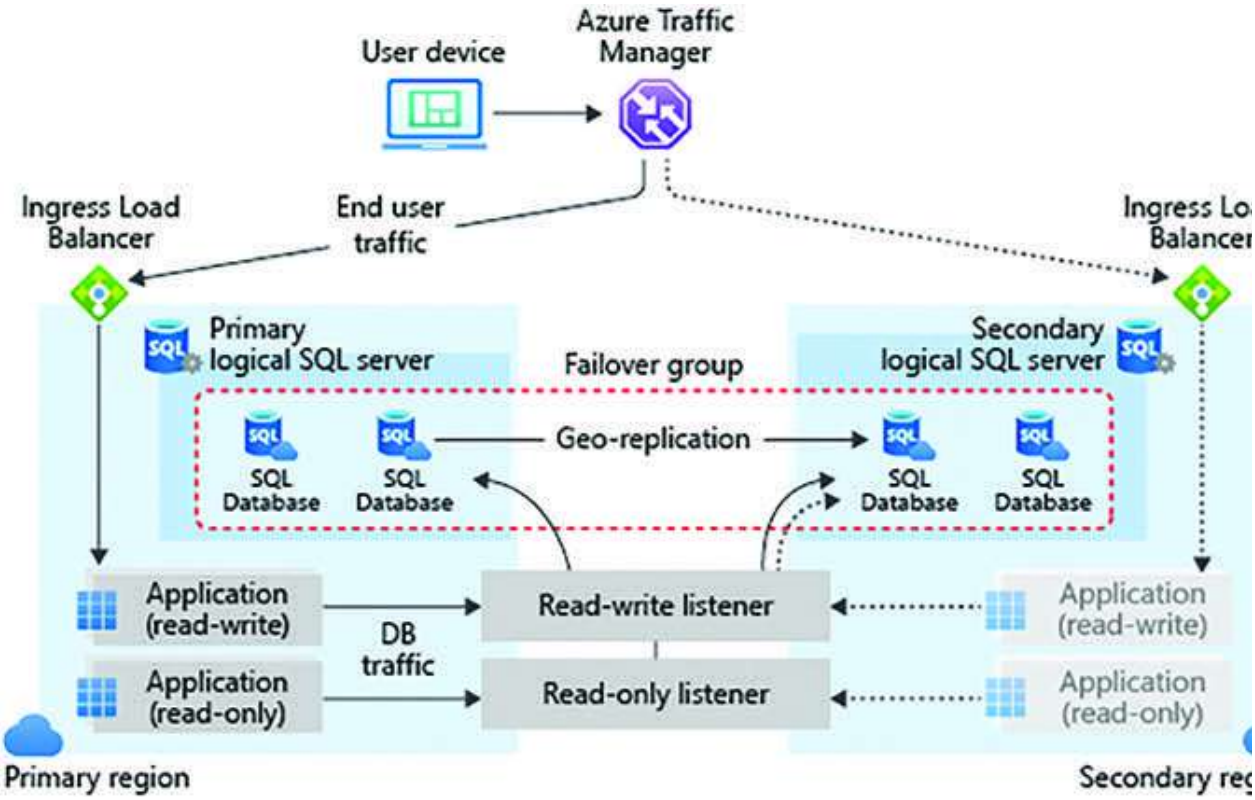


Figure 5.35: Auto Failover Groups for Azure SQL DB

As you might have noticed, Auto-failover group is basically a declarative abstraction on top of the active geo-replication feature, simplifying the management and deployments of geo-replicated databases as scale.

The same architecture for Azure SQL MI is also available and shown in Figure 5.36 as documented in the official Microsoft documentation where it is managing an entire instance instead of group of databases:

Auto-failover groups overview & best practices - Azure SQL Managed Instance | Microsoft Learn

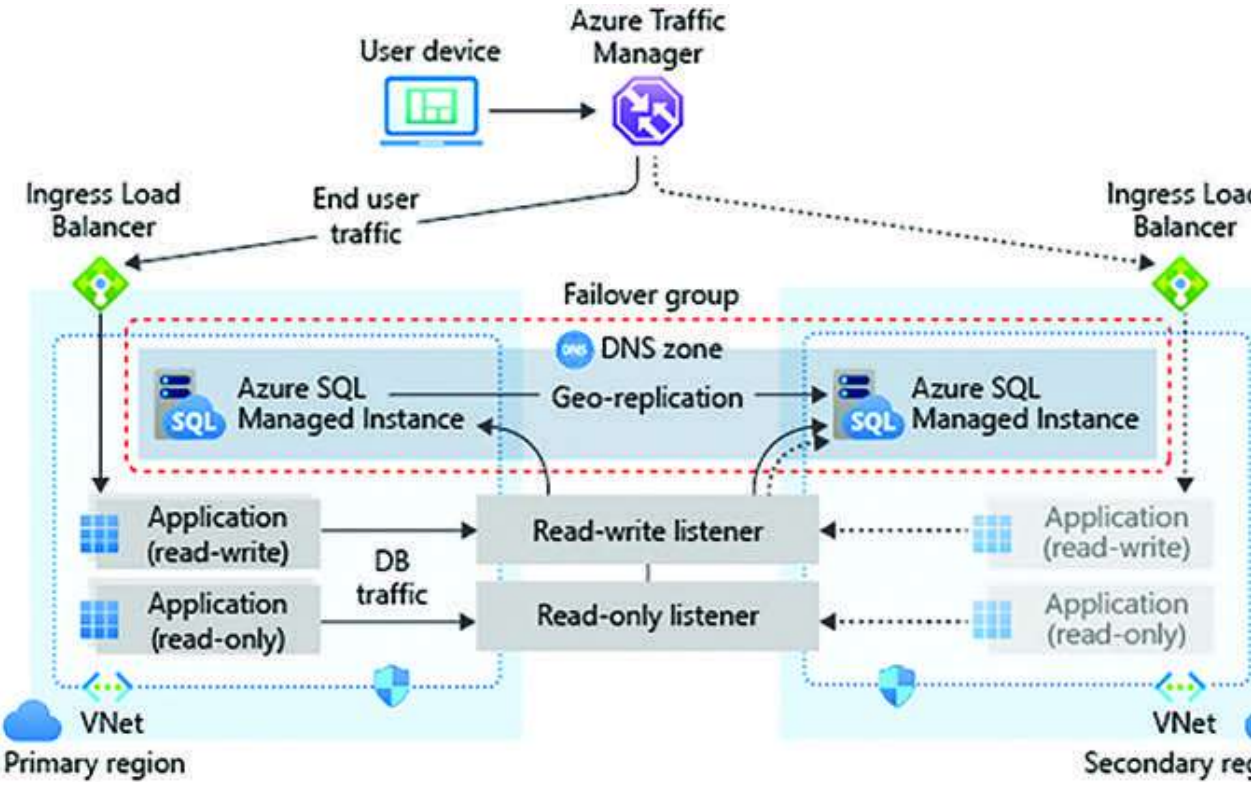


Figure 5.36: Auto Failover Groups for Azure SQL MI

It is highly recommended that when you work on creating the business continuity strategy for your Azure SQL DB and Azure SQL MI deployments,

you please refer to the official documentation to understand the target RPO, RTO requirements, best practices, and checklists for HA/DR requirements. You can start from this official documentation from Microsoft: Azure SQL Database high availability and disaster recovery checklist - Azure SQL Database | Microsoft Learn

After this, you can create your own plan for business continuity.

Azure SQL VM

The business continuity plan for Azure SQL Server Virtual Machine (VM) deployments supports all the previously studied High Availability Disaster Recovery (HADR) features as both Azure-only and/or hybrid solutions. In an Azure-only solution, the entire HADR feature runs in Azure. In a Hybrid solution, part of the solutions runs in Azure and the other part runs on-premises of your organization.

This gives you the ability to run partially or fully on the cloud, depending on your budget and business requirements. Because Azure SQL Server VM is more of an IaaS solution, the responsibility of configuring the HADR features is completely owned by the user.

Azure SQL Server VMs support all the previously studied HADR features

Always On availability groups

Always On Failover cluster instances (FCIs)

Log Shipping

SQL Server backup and restores with Azure Blob storage.

To know more about how you can deploy Azure only or Hybrid solutions with each of these HADR options, please refer to the Microsoft

documentation available on this topic here at: High availability, disaster recovery, business continuity - SQL Server on Azure VMs | Microsoft Learn

With this, we come to the end of this chapter. You have learned about the different HADR features in SQL Server and also the options available in the Azure SQL family.

Conclusion

In conclusion, we explored a range of topics essential to SQL Server's business continuity strategy. This included an introduction to fundamental concepts such as log shipping, Always On availability groups, and Failover Clustered Instances, applicable to both Windows and Linux versions of SQL Server. Additionally, we delved into the comparative analysis of SQL Server's clustering on Linux and Windows, highlighting their respective similarities and differences. Building upon these core principles, we expanded our knowledge to encompass the various High Availability/Disaster Recovery (HA/DR) strategies employed within the SQL on Azure ecosystem, rounding out our comprehensive study of the subject.

The preceding sections provided insights into various SQL solutions and their respective high availability features. Should you be considering a database migration, the forthcoming chapter will serve as a guide to streamline and expedite your migration process. It will concentrate on the strategic planning and implementation of the migration.

Additional Resources

Windows Server Failover Cluster (WSFC)

DH2i DxEnteprise

HPE Serviceguard

Pacemaker

Mssql-server-ha

Create a failover cluster | Microsoft Learn

Create new Failover Cluster Instance - SQL Server Always On | Microsoft Learn

Adding or remove nodes in a SQL Server Failover cluster setup

DH2i: HA Instances for Linux | DH2i Support Portal

Workload for Microsoft SQL Server Always on Failover Instance | HPE Serviceguard Solutions for Microsoft SQL Server for Linux User Guide

Configure Failover Clustered Instance for SQL on Linux

ClusterLabs 1. Introduction — Clusters from Scratch

Configure RHEL FCI for SQL Server on Linux - SQL Server | Microsoft Learn

Database Mirroring for SQL Server

SQL Server business continuity

SQL Server Linux availability group- high availability

Use Availability group wizard SQL Server management studio

Application Registration in Microsoft Entra ID

Custom role for Fence Agent

Assign custom role to the service principal

Create the load balancer using the Azure portal

microsoft.sql role

SQL Server – The Ansible way! - Microsoft Community Hub

Configure the fence agent

mssql/README.md at master linux-system-roles/mssql (github.com)

mssql/README.md at master linux-system-roles/mssql (github.com)

serviceguard/ansible-sglx at Stable-v1.2 · HewlettPackard/serviceguard (github.com)

Deploy SQL Server containers on Kubernetes with StatefulSets - SQL Server | Microsoft Learn

Deploy availability groups with DH2i DxEnterprise on Kubernetes - SQL Server | Microsoft Learn

Configure SQL Server Always On availability group on Windows and Linux - SQL Server | Microsoft Learn

Availability Groups with Mixed OS | DH2i Support Portal

What is a distributed availability group - SQL Server Always On | Microsoft Learn

Contained Databases - SQL Server | Microsoft Learn

long term backup retention

Automatic, geo-redundant backups - Azure SQL Database | Microsoft Learn

Restore a database from a backup - Azure SQL Database | Microsoft Learn

Tutorial: Geo-replication & failover in portal - Azure SQL Database | Microsoft Learn

Active geo-replication - Azure SQL Database | Microsoft Learn

Auto-failover groups overview & best practices - Azure SQL Managed Instance | Microsoft Learn

Azure SQL Database high availability and disaster recovery checklist - Azure SQL Database | Microsoft Learn

High availability, disaster recovery, business continuity - SQL Server on Azure VMs | Microsoft Learn

CHAPTER 6

Accelerate SQL Server Modernization

Introduction

In today's rapidly evolving data landscape, businesses are increasingly reliant on their data infrastructure to drive innovation, make informed decisions, and gain a competitive edge. SQL Server 2022 and Azure SQL offerings represent a significant milestone in the world of data management, offering a wide array of modernization features and enhancements that empower organizations to harness the full potential of their data. Modernizing data estate is a strategic move that empowers organizations to harness the full potential of their data while ensuring scalability, security, and cost efficiency. Though the question for most organizations remains this: Is it the right time to modernize? Well, the answer to this question is multifold, but it does raise a few more questions for the stakeholders to consider:

Do we need to modernize to reduce costs?

Do we need to modernize to remain compliant?

We need to keep vendor support intact; should we modernize?

As an ISV, should there be an update to enable more choices and features for customers?

Should we separate application modernization cycles from data modernization cycles?

One of the most important steps in modernization of data estate is migration of databases and applications using migration strategies tailored to your organization's needs. In terms of database modernization, whether you are moving from on-premises SQL Server to Azure SQL, upgrading within the SQL Server family, undergoing a heterogeneous migration, from say, Oracle to the SQL family, or adopting hybrid scenarios, each scenario represents a unique challenge, and relevant strategies must be adopted to make the migration effort successful.

Structure

In this chapter, we will cover the following topics:

Database Migration

Importance of Database Migration

Migration Patterns

Migration Process Flow

Upgrade SQL Server to SQL Server 2022

Benefits of Cloud Databases

Phases of Migration

Database Migration

Heterogeneous Database Migration

Database Migration

Database migration is the process of moving a database from one environment or system to another. This can involve changing the database software, moving to new hardware, or transferring the database to a cloud environment. Database migration can be complex and may include changes in database structure, data types, and the way data is accessed or used.

Types of Database Migration

Platform Migration:

Moving a database from one database management system (DBMS) to another (example, from Oracle to Microsoft SQL Server). This often involves significant changes in database schemas, SQL queries, and application code to accommodate differences between the systems.

Hardware Migration:

Transferring a database from one physical server or storage system to another. This might be necessary due to hardware upgrades, consolidation of servers, or moves to more powerful infrastructure.

Cloud Migration:

Moving a database from on-premises servers to a cloud-based environment. This can include migrating to a cloud-based DBMS (example, Azure SQL Database) or simply hosting the existing DBMS on cloud infrastructure.

Version Upgrade:

Upgrading the DBMS to a newer version may involve changes in database features, performance optimizations, and compatibility considerations.

Schema Migration:

Involves changes to the database schema, such as adding, modifying, or removing tables, columns, indexes, or relationships. Schema migrations are often part of broader application upgrades or refactoring efforts.

Database migration is a critical task that requires careful planning, execution, and testing to ensure success. Whether upgrading hardware, moving to the cloud, or switching DBMS platforms, understanding the intricacies of database migration is essential to minimize risk and achieve desired outcomes.

Importance of Database Migration

Database migration is an essential process in the lifecycle of IT systems and infrastructure. The importance of database migration can be highlighted through several key factors:

Improved Performance and Scalability:

Optimization: Migrating to a more modern or powerful database platform can significantly improve system performance. This includes faster query processing, improved data retrieval speeds, and the ability to handle larger datasets.

Scalability: Database migration allows businesses to scale their database infrastructure to meet growing demands, whether by upgrading hardware, moving to a cloud environment, or adopting a more scalable DBMS.

Cost Efficiency:

Reduced Costs: Migrating to a cloud-based or more cost-effective database can lower operational expenses by reducing the need for expensive on-premises hardware and maintenance.

License Savings: Switching to a different DBMS can lead to savings on software licensing costs, especially if moving to open-source or subscription-based models.

Access to Advanced Features:

New Capabilities: Newer database platforms often come with advanced features such as improved data analytics, machine learning integration, and enhanced security protocols.

Better Integration: Migrating to a modern database platform can facilitate better integration with other technologies and applications, enabling more efficient workflows and data sharing.

Enhanced Security and Compliance:

Improved Security: Migration offers an opportunity to upgrade to a database system with better security features, such as encryption, access controls, and audit logging.

Regulatory Compliance: Adhering to new or evolving regulatory requirements may necessitate a database migration to ensure compliance with standards such as GDPR, HIPAA, or PCI-DSS.

Consolidation and Centralization:

Simplification: Consolidating multiple databases into a single, centralized system can simplify data management, reduce redundancy, and improve data consistency.

Easier Maintenance: Centralized databases are easier to manage and maintain, reducing the complexity of updates, backups, and disaster recovery efforts.

Business Continuity and Disaster Recovery:

Enhanced Recovery Options: Migrating to a platform with better disaster recovery features, such as automated backups, replication, and failover capabilities, can improve business continuity.

Minimized Downtime: Database migration can help reduce the risk of downtime by moving to a more resilient infrastructure or environment.

Future-Proofing the Infrastructure:

Staying Current: As technology evolves, older database systems may become obsolete or unsupported. Migration ensures that the organization remains on a platform that is actively maintained and updated.

Adaptability: Migrating to a flexible and adaptable database system prepares the organization for future technological changes, making it easier to adopt new innovations and trends.

Enhanced User Experience:

Improved Application Performance: Faster and more reliable databases enhance the overall user experience by providing quicker access to data and reducing application latency.

New Functionalities: Migration can enable new functionalities within applications, offering users more features and capabilities.

Support for Digital Transformation:

Enabling Innovation: As businesses undergo digital transformation, database migration is often necessary to support new digital initiatives, such as big data analytics, artificial intelligence, and IoT.

Modernization: Migrating to modern database platforms aligns with broader efforts to modernize IT infrastructure and support innovative business models.

Strategic Business Goals:

Global Expansion: Organizations expanding globally may need to migrate databases to support multiple geographic regions, comply with local regulations, and provide better performance to users worldwide.

Mergers and Acquisitions: Database migration is often required during mergers and acquisitions to integrate systems and unify data across the merged entities.

Reducing Technical Debt:

Eliminating Legacy Systems: Migrating away from outdated and inefficient databases reduces technical debt, lowers the risk of system

failures, and improves long-term maintainability.

Simplified Architecture: By moving to a more streamlined and modern database system, organizations can simplify their overall IT architecture, making it easier to manage and evolve.

Database migration is critical for organizations seeking to improve performance, reduce costs, enhance security, and stay competitive in a rapidly evolving technological landscape. It not only ensures that the database infrastructure is aligned with current business needs but also prepares the organization for future growth and challenges.

Migration Patterns

Of course, modernization of data estate involves multiple activities such as application code migration, re-architecture, compatibility check between application and database, incorporating new drivers, and so on. In the context of this topic, strategies for migration fall into four patterns: rehost, refactor, rearchitect, or rebuild. Different organizations adopt different strategies depending on various business drivers and migration goals. There are many patterns that organizations may adopt; for example, non-critical applications would rehost while highly complex and business-critical applications are needed to be rearchitected. Let us look at these patterns in detail, which are applicable for both on-premises and the Azure cloud platform:

platform:

platform: platform:

platform: platform:

platform: platform: platform: platform: platform: platform: platform: platform: platform: platform: platform: platform: platform: platform:

platform: platform:

platform: platform:

Table 6.1: Migration Patterns

Many a time, modernization and migration are used interchangeably; however, migration is a subset or one of the steps within a modernization effort. Migration in SQL Server involves moving data from one SQL Server instance to another or to Azure SQL offerings. This could be for various reasons, such as upgrading to a newer version, moving to different hardware, or migrating to the cloud. While modernization involves updating prevailing applications with newer computing approaches and

frameworks, hardware, or use of cloud-native technologies. This can be achieved in both on-premises environments using new technology or platform stacks or by using PaaS services such as Azure SQL databases with some added benefits. In the rest of this chapter, we will focus specifically on migration of homogeneous and heterogeneous databases to the SQL platform, both on-premises and Azure cloud data services.

Migration Process Flow

The migration process flow is a series of steps that you need to follow to move your data and applications from one environment to another, such as from on-premises to the cloud. The migration process flow can vary depending on the complexity, scale, and scope of your migration project, but it generally consists of five broad phases: discover, assess, migrate, cutover, and optimize.

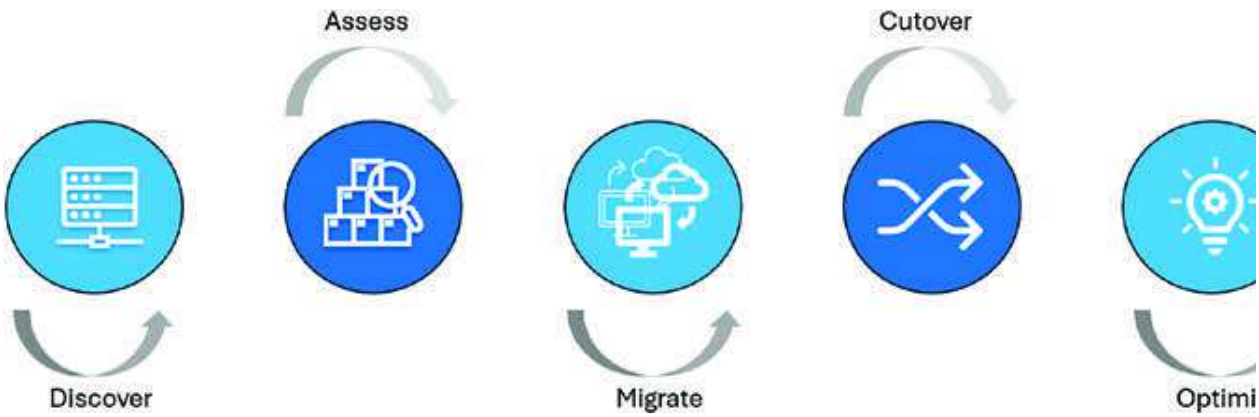


Figure 6.1: Migration Process Flow

Pre-Migration Phase

Discover: In this phase, an effort has to be made to identify and inventory the assets that need to be migrated, such as servers, databases, applications, and dependencies. This phase helps to understand the current state of the source environment and the requirements for migration.

Assess: In this phase, analysis of the feasibility, cost, and benefits of migrating the assets is done. Different migration strategies are then evaluated, such as lift and shift, lift and optimize, or rearchitect or rebuild. Also, any potential issues or risks that might affect the migration process or outcome must be evaluated at this time. This phase helps in planning and prioritizing the migration activities and selecting the best migration tools for the migration scenario.

Migration Phase

Migration: In this phase, the migration plan is executed with the help of the selected migration tools to replicate, transfer, and validate the data. Monitoring and troubleshooting the migration process is an important activity to ensure that it runs smoothly and successfully. This phase helps in moving data assets with minimal downtime and disruption.

Post Migration Phase

Cutover: In this phase, the production traffic is switched over from source environment to target environment. The final checks are performed, and connectivity tests are conducted to verify that everything is working as expected. A few additional steps, such as updating DNS records, firewall rules, and connection strings to point to the new environment, must be taken into consideration.

Optimize: In this phase, fine-tuning of the target environment is performed after the migration is complete. Optimization of the target environment resources for performance, availability, security, and cost are performed. All the best practices for governance, management, and operations must be part of the cookbook for this phase. This phase helps maximize the value and benefits of migrating to the new environment.

Upgrade SQL Server to SQL Server 2022

Before we delve into upgrade, we must understand the difference between an upgrade and a migration. Upgrade happens within the same family of products, from the legacy version to the newest version of the product. Upgrade typically is a straightforward process, for which tools are typically provided by the software vendor. For example, moving SQL Server 2017 to SQL 2019 is an upgrade process. While migration is a process of changing the platform: software or hardware. For example, moving from Windows to Linux or from a physical server to a virtual machine would be a migration.

Upgrades are categorized into three categories: in-place upgrade, side-by-side upgrade, and rolling upgrade. The difference between these categories is mentioned in Table

SQL Server 2022 provides multiple upgrade paths from legacy SQL Server instances. Microsoft supports upgrade instances of SQL Server 2012, SQL Server 2014, SQL Server 2016, SQL Server 2017, or SQL Server 2019 directly to SQL Server 2022. For SQL Server 2008 and R2, the organizations must plan for a side-by-side upgrade, or a migration, to move to SQL Server 2022, as there is no common overlap between a supported mainstream operating system.

system. system.

system. system. system. system. system. system. system. system. system. system. system. system. system. system.
system. system. system. system. system. system. system. system. system. system. system. system. system.
system. system. system. system. system. system. system. system. system. system. system. system.

system. system. system. system. system. system. system. system. system. system. system. system. system.
system. system. system. system. system. system. system. system. system. system. system. system.
system. system. system. system. system. system. system. system. system. system. system.
system. system. system. system. system. system. system. system. system. system. system. system.

Table 6.2: Categories of SQL Server upgrades

This section describes various steps involved in upgrading SQL Server Database Engine.

Preparation for Migration

To prepare for the migration, the following tools are required to be installed on any machine or virtual machine having access to the SQL

Server instance to be upgraded:

Latest version of the MAP Toolkit:

https://www.microsoft.com/en-us/download/details.aspx?id=7826

The MAP Toolkit is used for multi-product assessment and planning. It assesses a network environment using agentless data collection technologies to gather inventory and performance information. Then provides assessment reports to aid organizations with their IT infrastructure planning.

Latest version of Data Migration Assistant:

https://www.microsoft.com/en-us/download/details.aspx?id=53595

The Data Migration Assistant (DMA) helps in upgrading to a modern data platform by detecting compatibility issues that have the potential to impact database functionality after the upgrade to a new version of SQL Server or to Azure SQL Database. DMA is a great tool that allows migrating schema, data, and uncontained objects from the source SQL Server to the target SQL Server based platform.

Latest version of the Database Experimentation Assistant:

https://www.microsoft.com/en-us/download/details.aspx?id=54090

Database Experimentation Assistant (DEA) is an experimentation solution for SQL Server upgrades. DEA can help you evaluate a targeted version of SQL Server for a specific workload. Customers upgrading from earlier versions of SQL Server (starting with 2005) to more recent versions of SQL Server can use the analysis metrics that the tool provides.

Pre-Migration Phase

In this phase, you must confirm the source SQL Server instance is supported to be migrated to SQL Server 2022 and you have addressed any prerequisites. In this phase, an inventory of the databases to be migrated is made. This is followed by assessing the databases for potential migration issues or blockers and resolving any issues that are uncovered.

Discover

The discover stage helps collate all the data sources and details about the features that are in use by existing SQL Server instances. It gives a better understanding of and helps in planning for the migration. The discover phase scans the network to identify all SQL Server instances along with their version and features in use. The MAP Toolkit helps in creating an inventory of SQL instances.

The following are the high level steps to use the MAP Tool Kit:

Download the MAP Toolkit, and then install it.

Run the MAP Toolkit.

Open the MAP Toolkit, and then on the left pane, select Database.

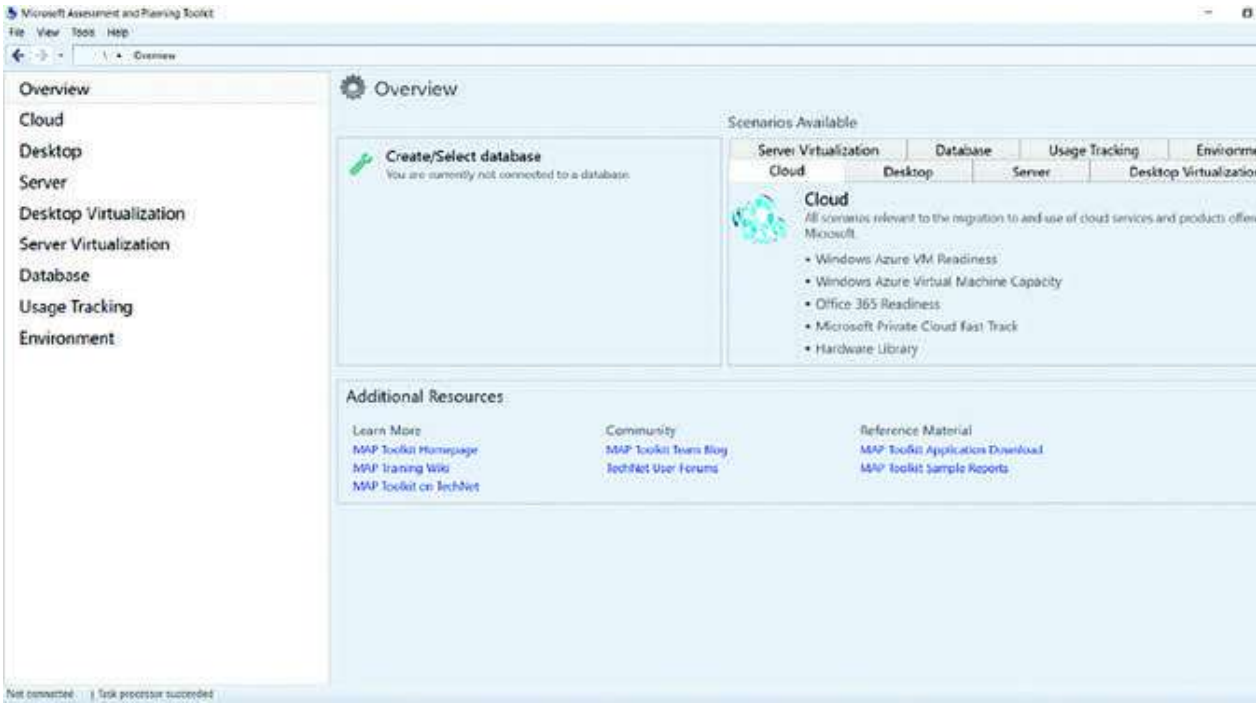


Figure 6.2: MAP Tool Kit

Select Create/Select

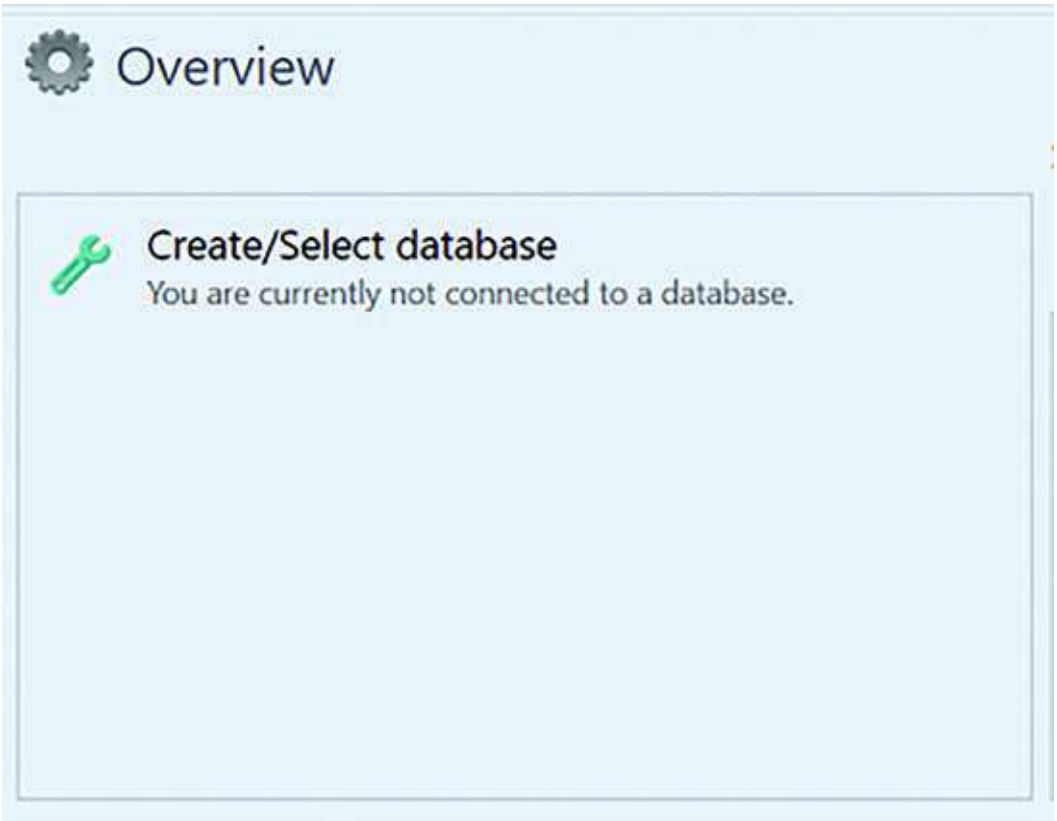


Figure 6.3: MAP Tool Kit Overview

Now, create an inventory database by selecting Create an inventory database and then enter a name for the database, a brief description, and then select OK.

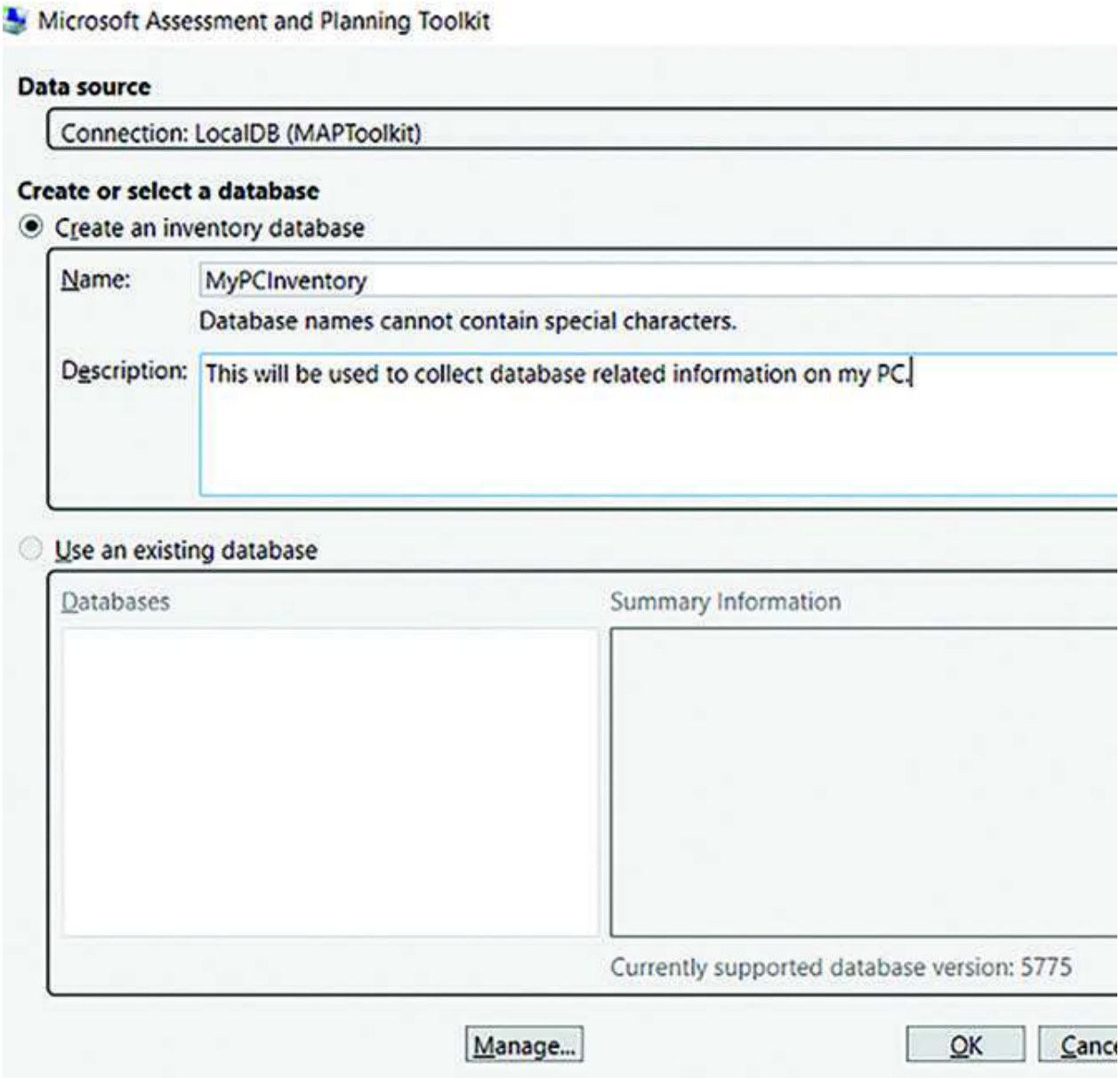


Figure 6.4: MAP Tool Kit Connection Settings

Select Collect inventory data to proceed:

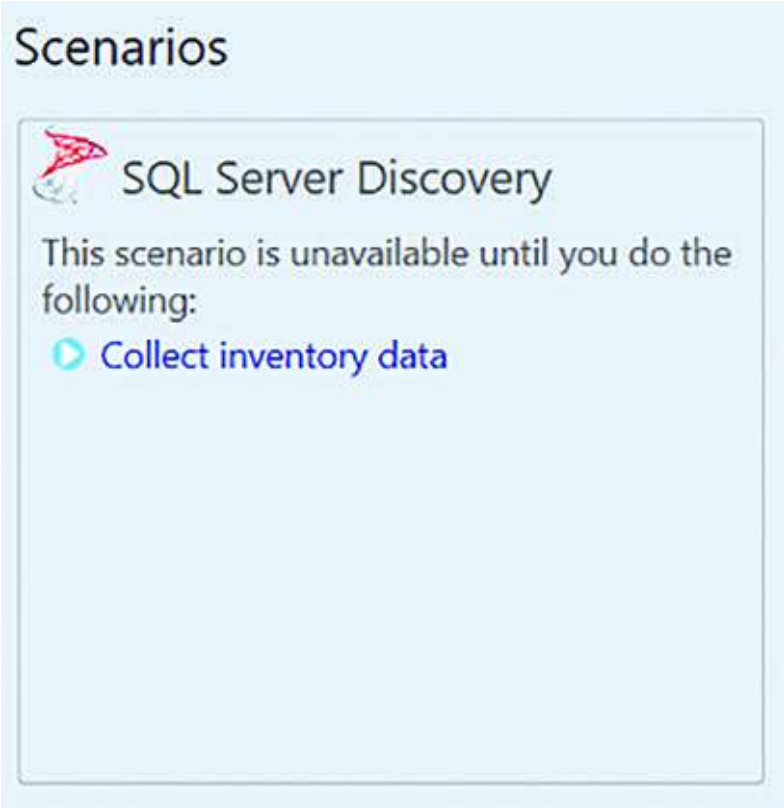


Figure 6.5: MAP Tool Kit SQL Server Discovery

In the Inventory and Assessment select SQL Server and SQL Server with Database and then select

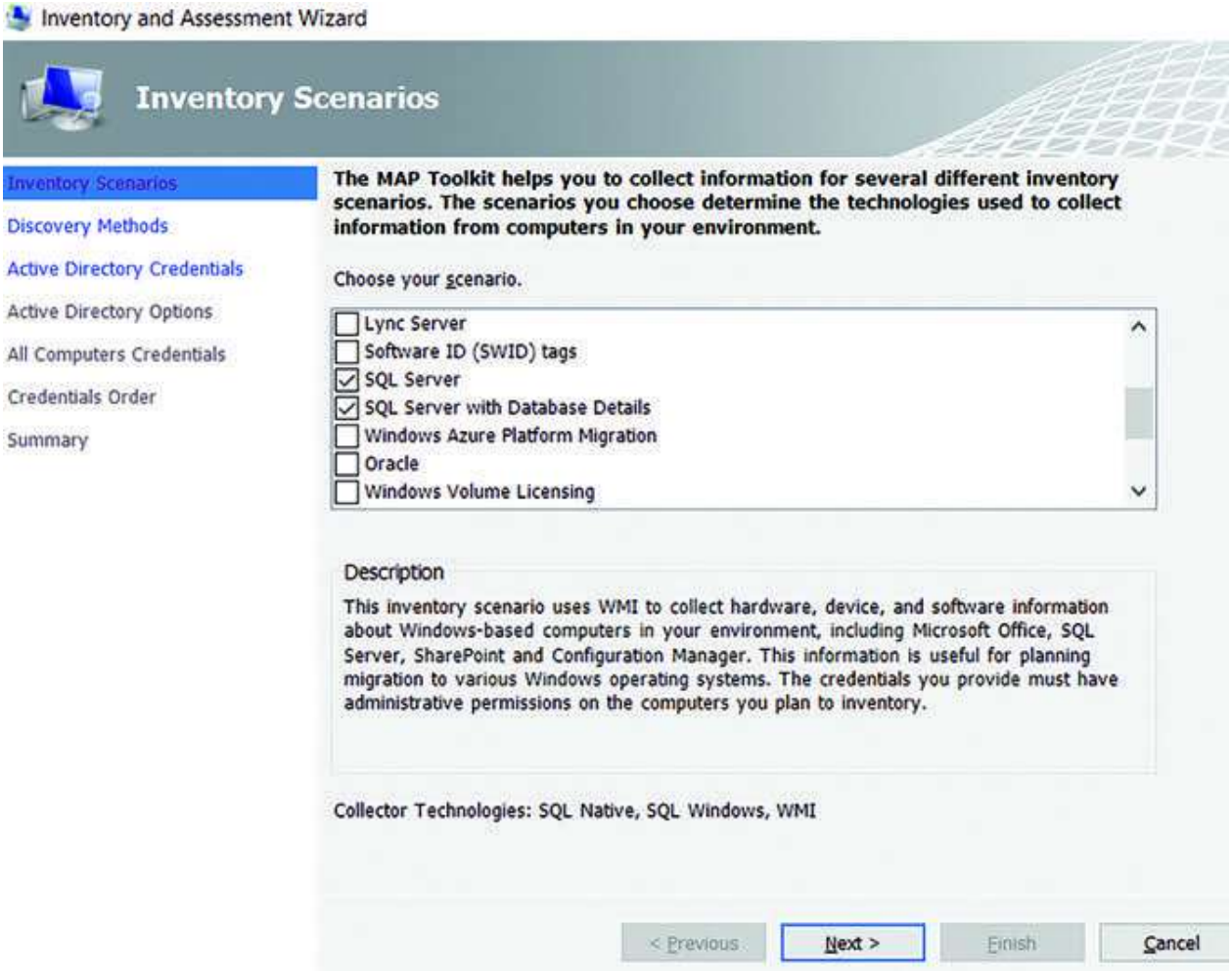


Figure 6.6: MAP Tool Kit Inventory Scenarios

Select the method to search the machines on which the SQL Server instances are hosted, and then select

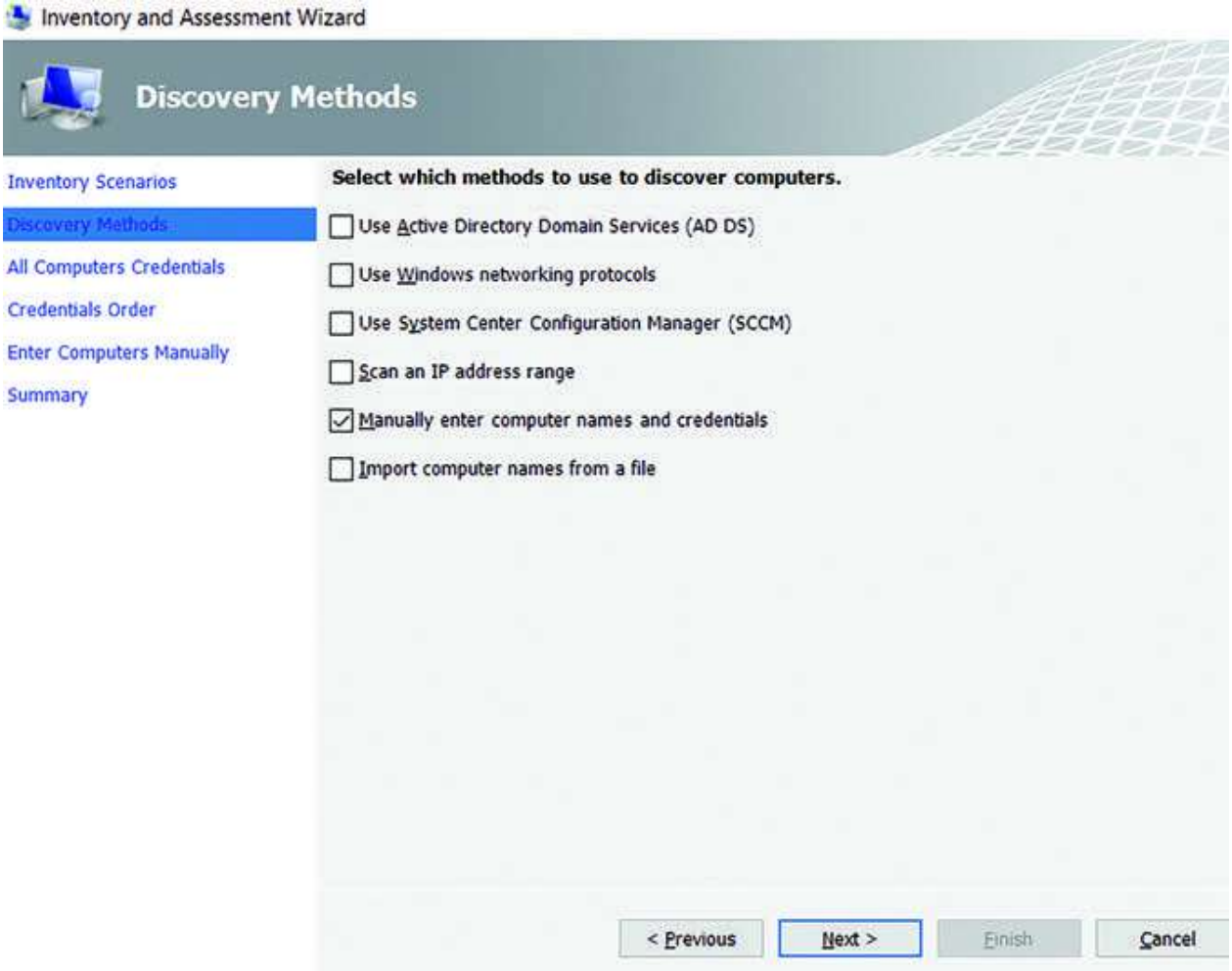


Figure 6.7: MAP Tool Kit SQL Server Discovery Methods

Next, enter the credentials or create new credentials for the SQL Server computers, and then select

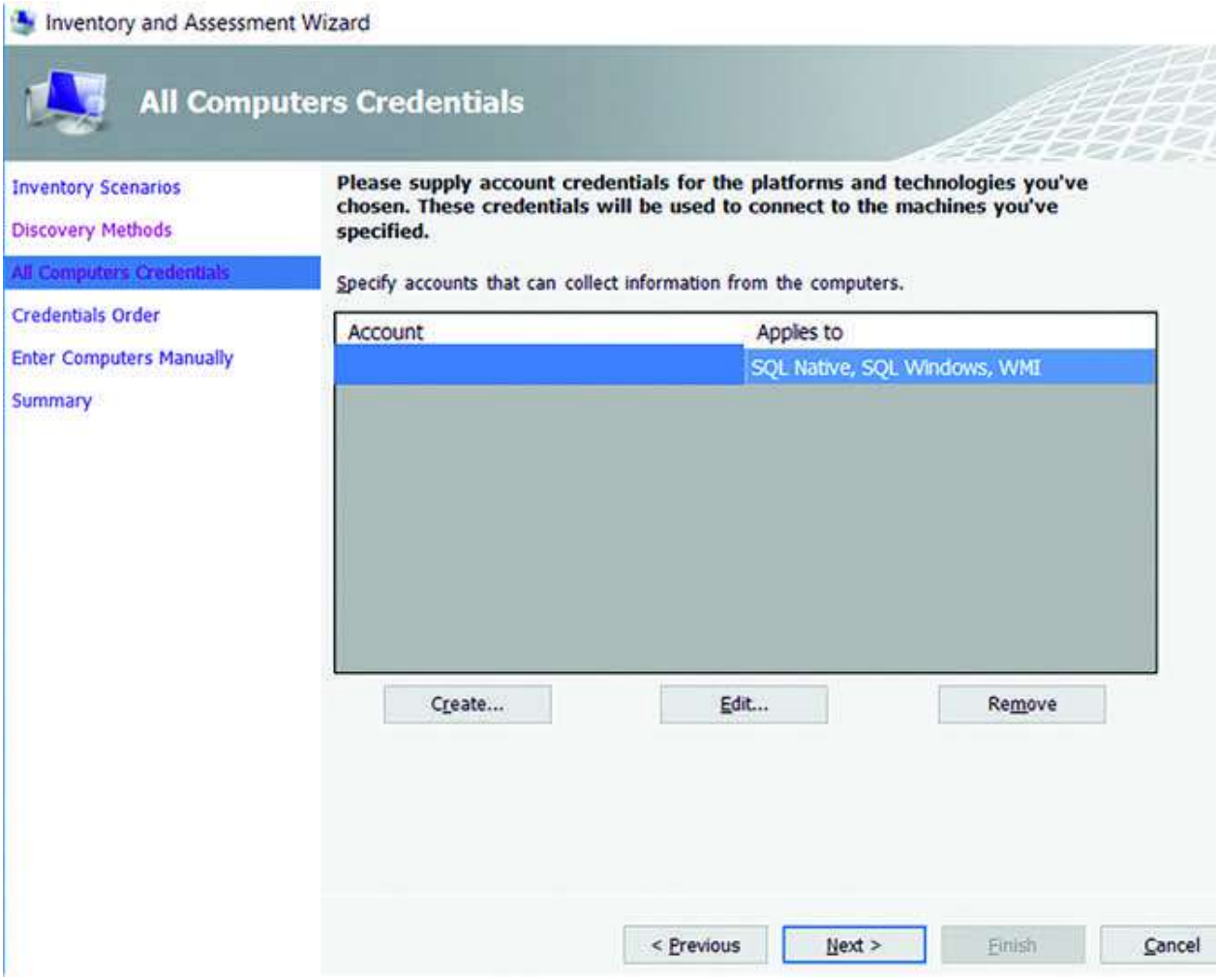


Figure 6.8: MAP Tool Kit Credentials Input

Set the order of the credentials, and then select

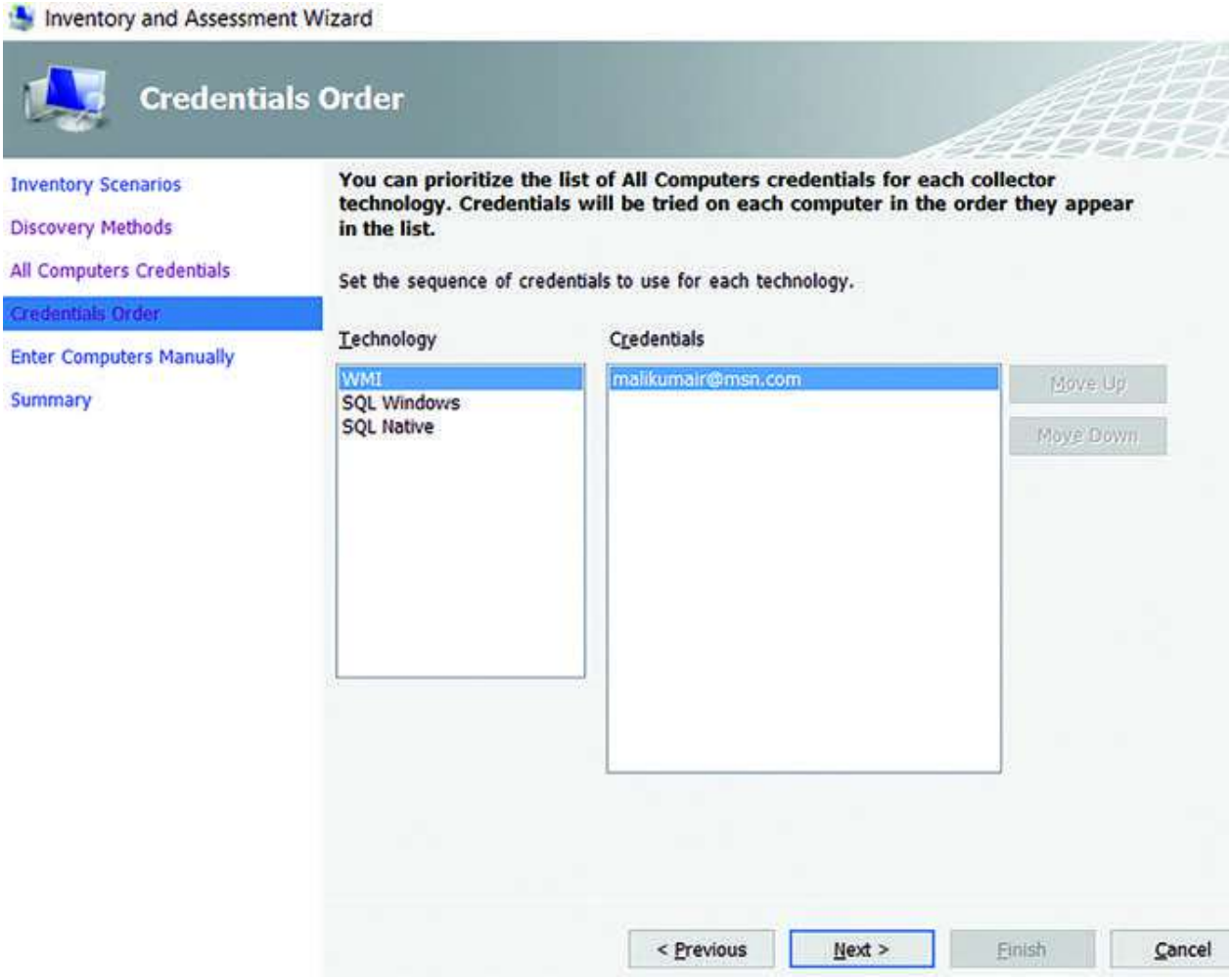


Figure 6.9: MAP Tool Kit Credentials Order

At this point, you need to specify the credentials for each SQL Server computer that you want to discover. You can use unique credentials for each machine, or you can choose to use the All Computer Credentials list.

After setting up the credentials, select and then click Next.

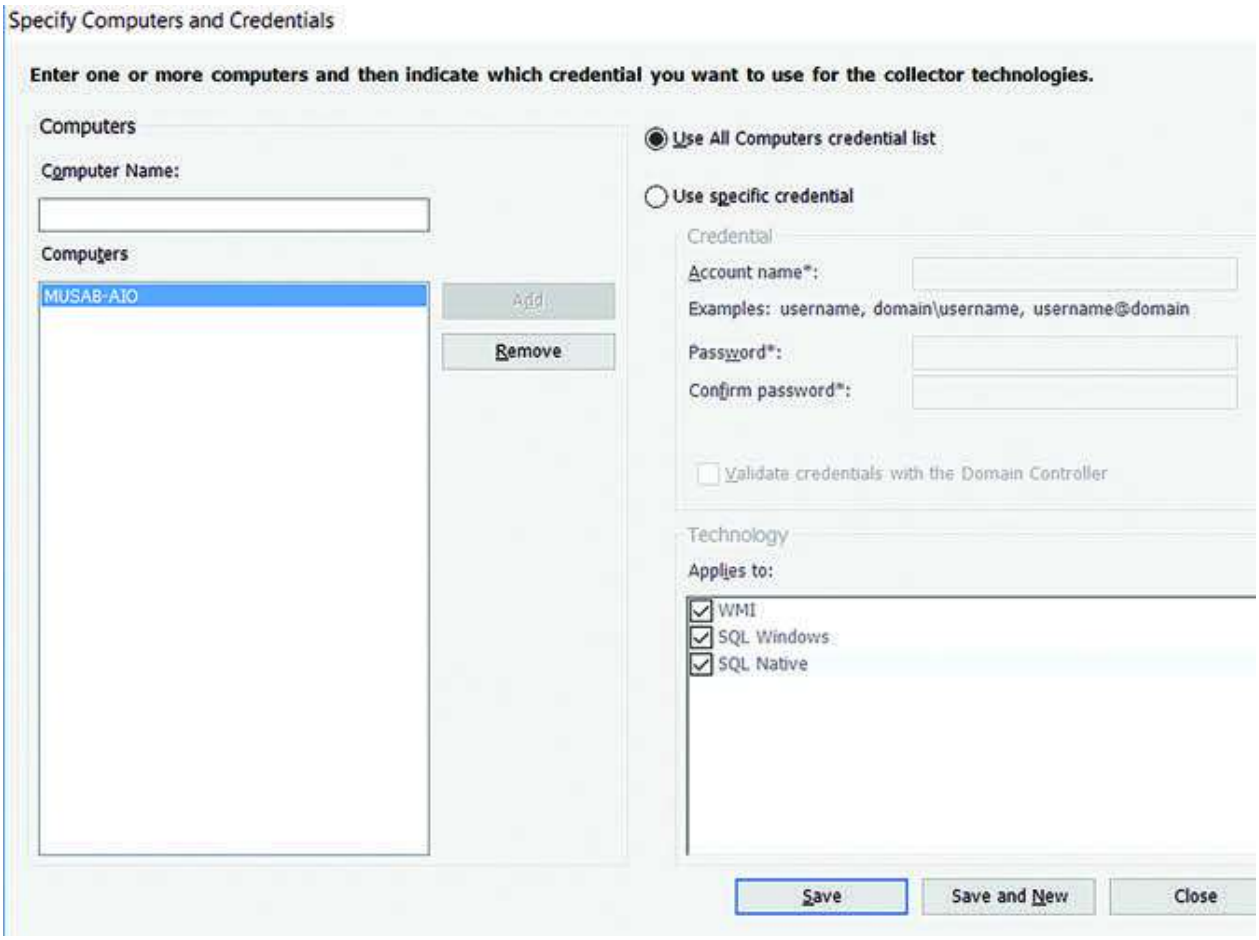


Figure 6.10: MAP Tool Kit Computers Selection

Now, you have to verify the selection and then click Finish.

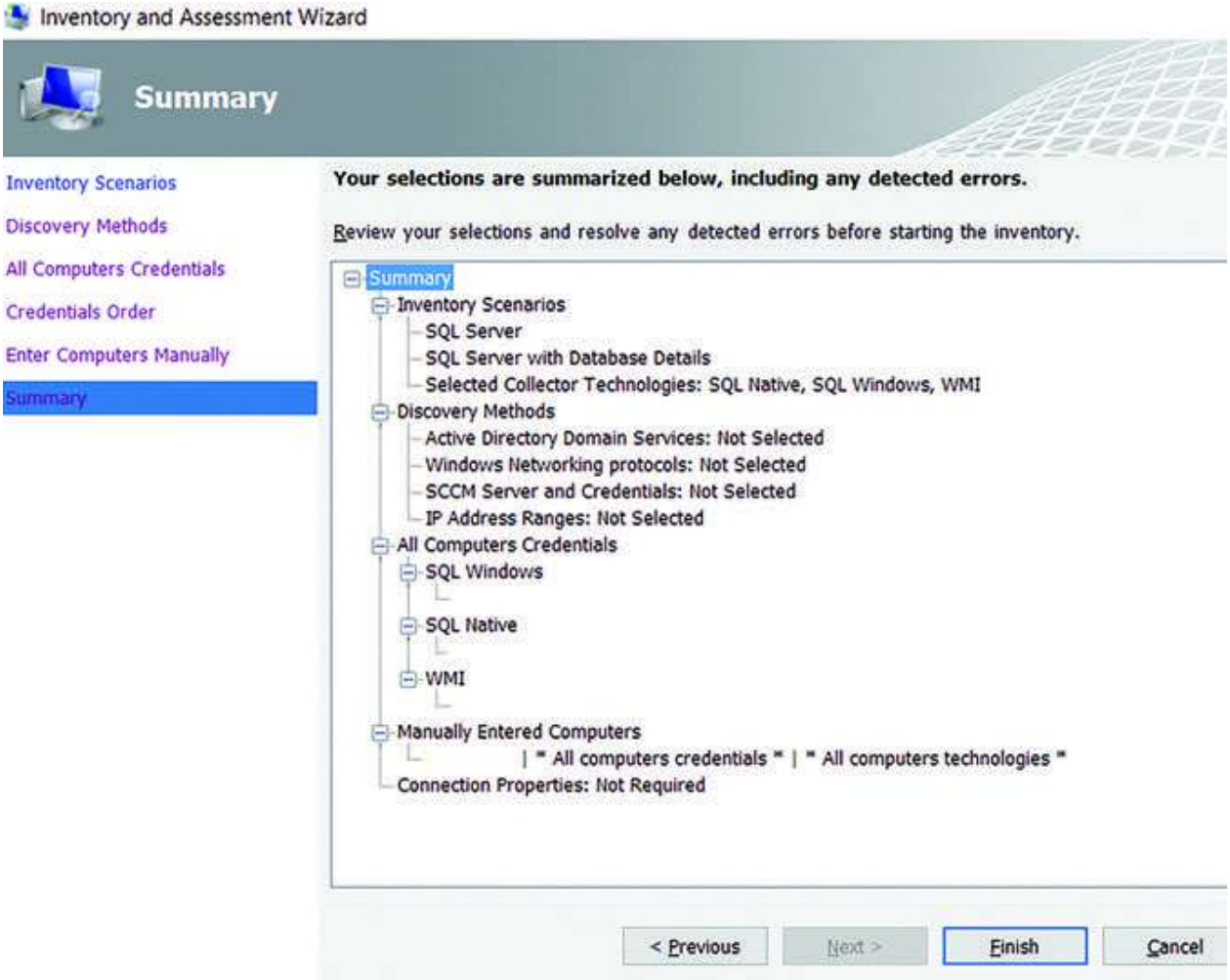


Figure 6.11: MAP Tool Kit Summary Page

After a few minutes (depending on the number of databases), the Data Collection summary report will be published.

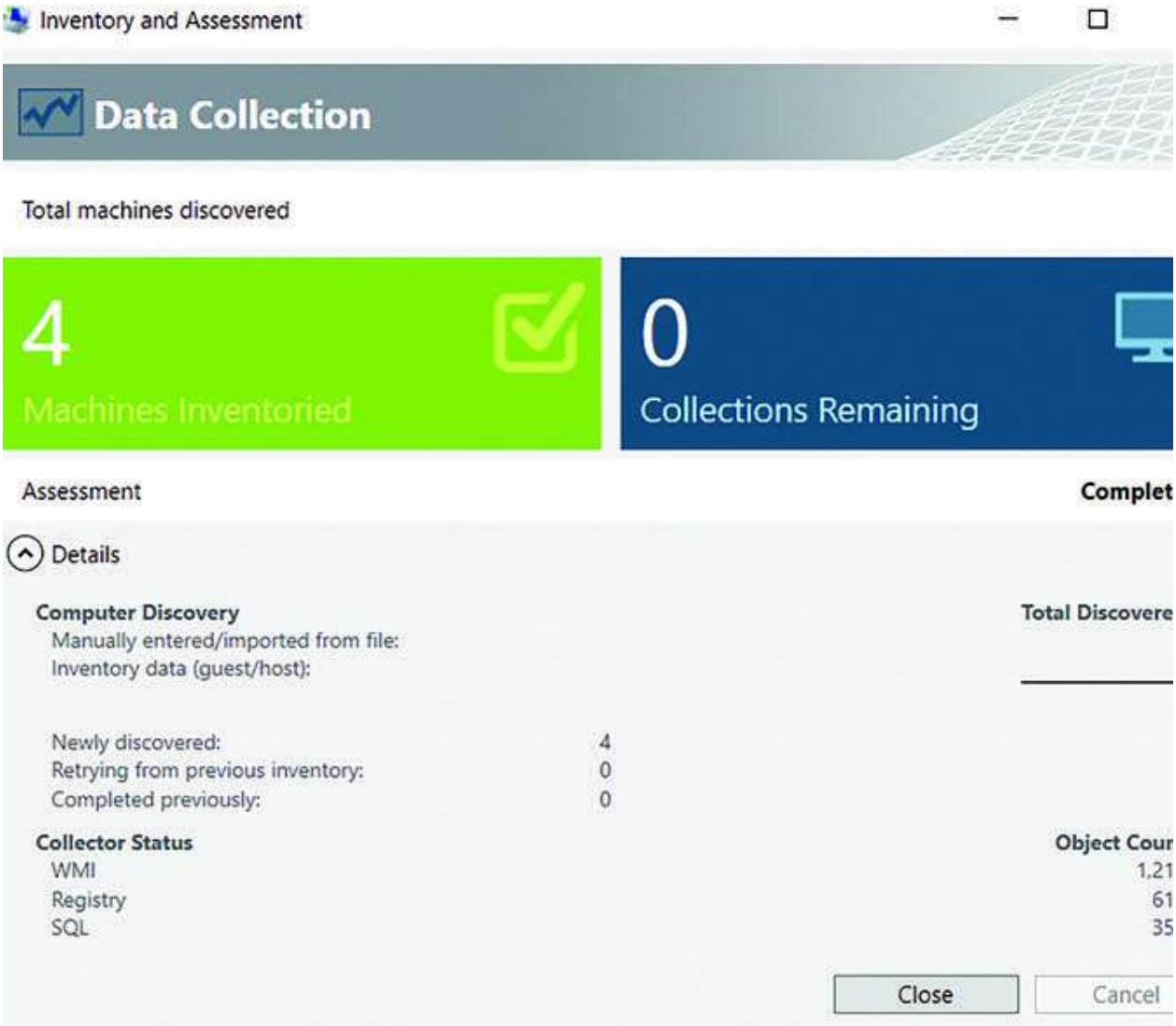


Figure 6.12: MAP Tool Kit Data Collection

Click on The Main window of the tool appears, showing a summary of the Database Discovery completed so far.

Report generation and data collection.

On the top-right corner of the tool, an Options page appears, which you can use to generate reports about the SQL Server Assessment and the Database Details.

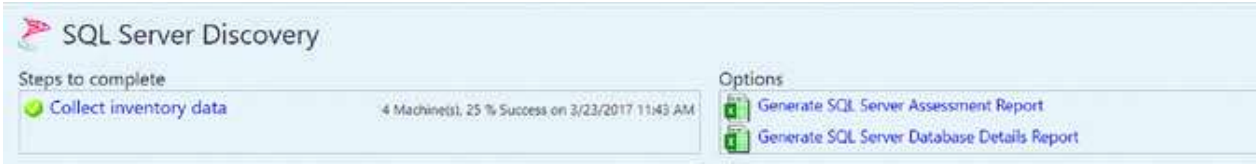


Figure 6.13: MAP Tool Kit SQL Server Discovery Report Generation

Select both options (one by one) to generate the report.

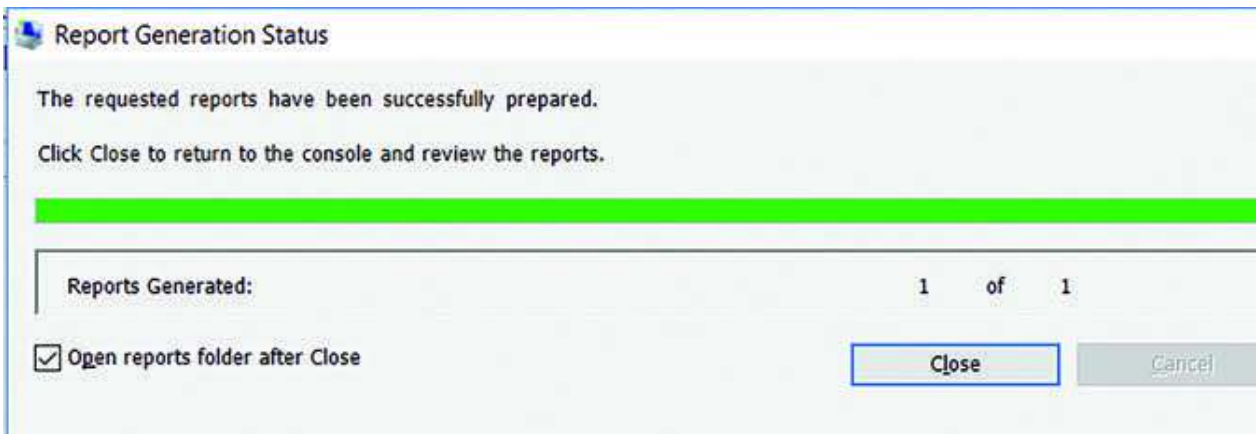


Figure 6.14: MAP Tool Kit Report Generation Satus

Assess and Convert

The next step in the migration journey is to assess the on-premises SQL Server instance(s). Data Migration Assistant (DMA) is one of the best tools available to assess source databases before upgrading SQL Server instance.

To use DMA to create an assessment, complete the following steps.

Download the DMA tool, and then install it.

Create a New Assessment project.

Select the New (+) icon, select Assessment under project type, and specify a project name. Then select SQL Server as the source and target both, and then click

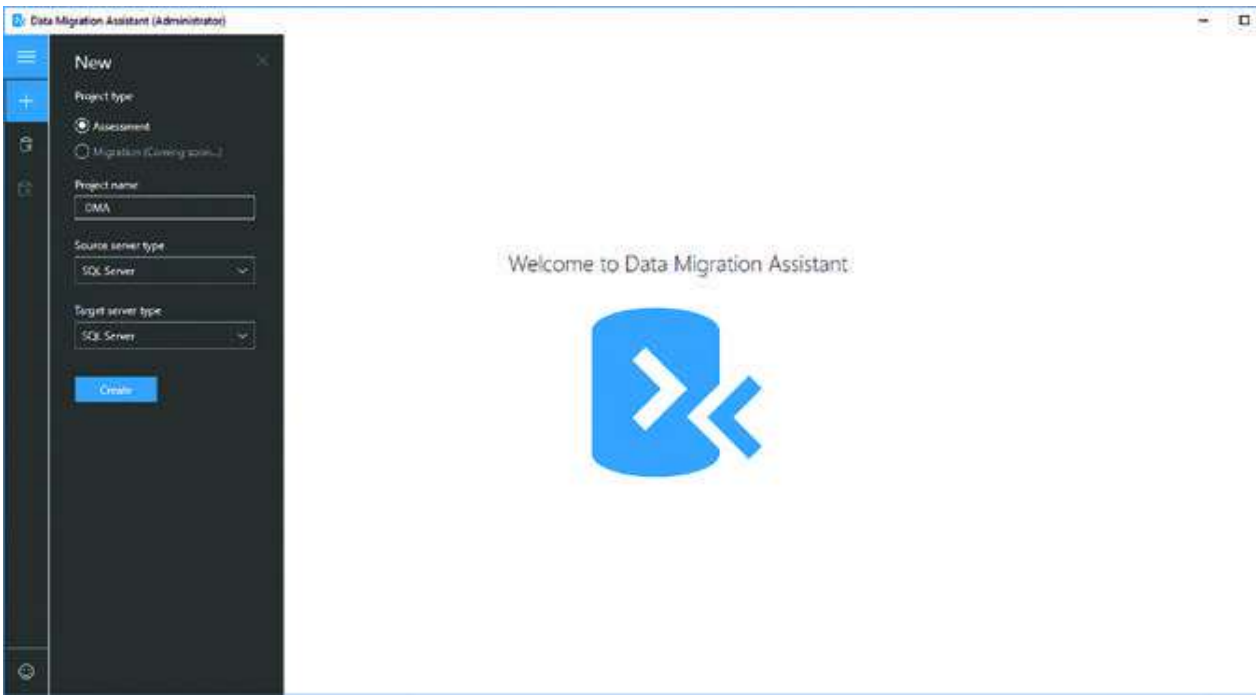


Figure 6.15: DMA Landing Page

Next, select the target SQL Server version that you plan to migrate to and against which this assessment will be executed. In this case, it must be SQL Server 2022 on Windows. After this, select assessment report types as Compatibility Issues, and then select

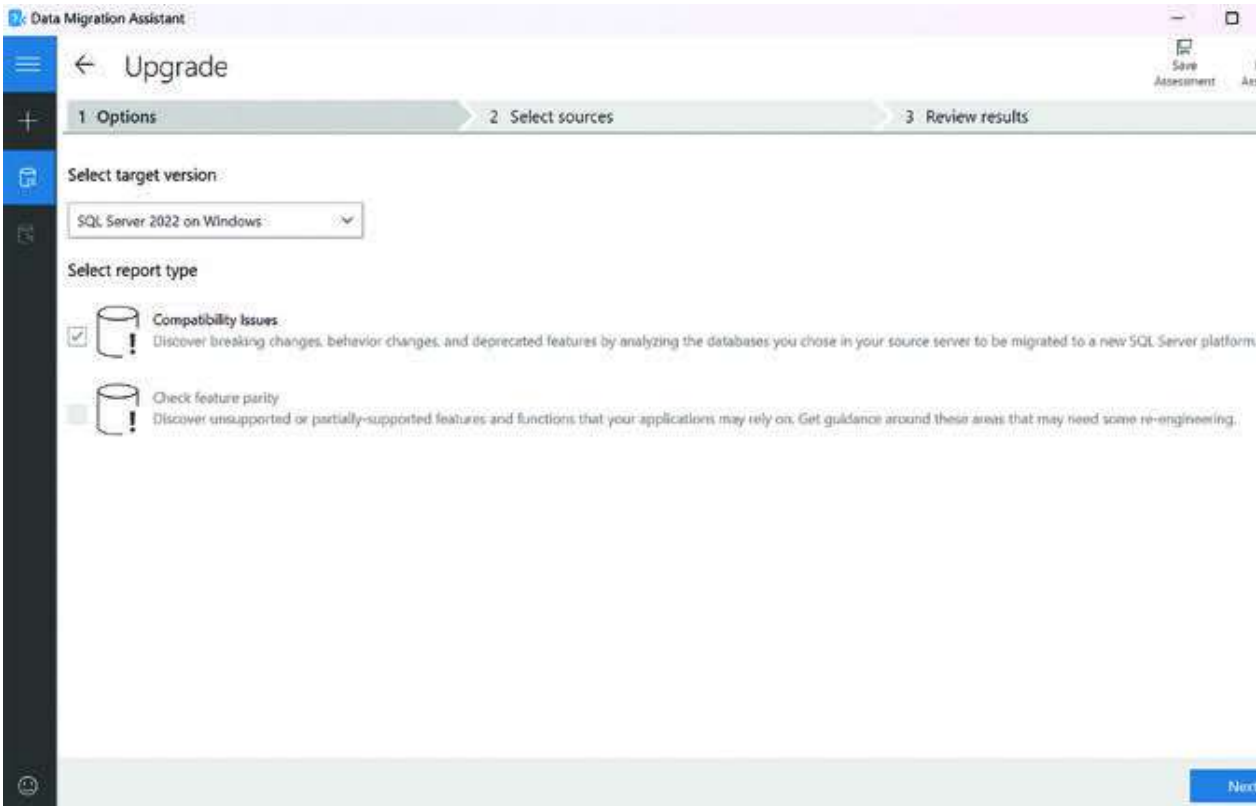


Figure 6.16: DMA Target Selection and Report Type

In Connect to a server specify the source SQL Server name, specify the Authentication type and Connection properties, and then click Connect.

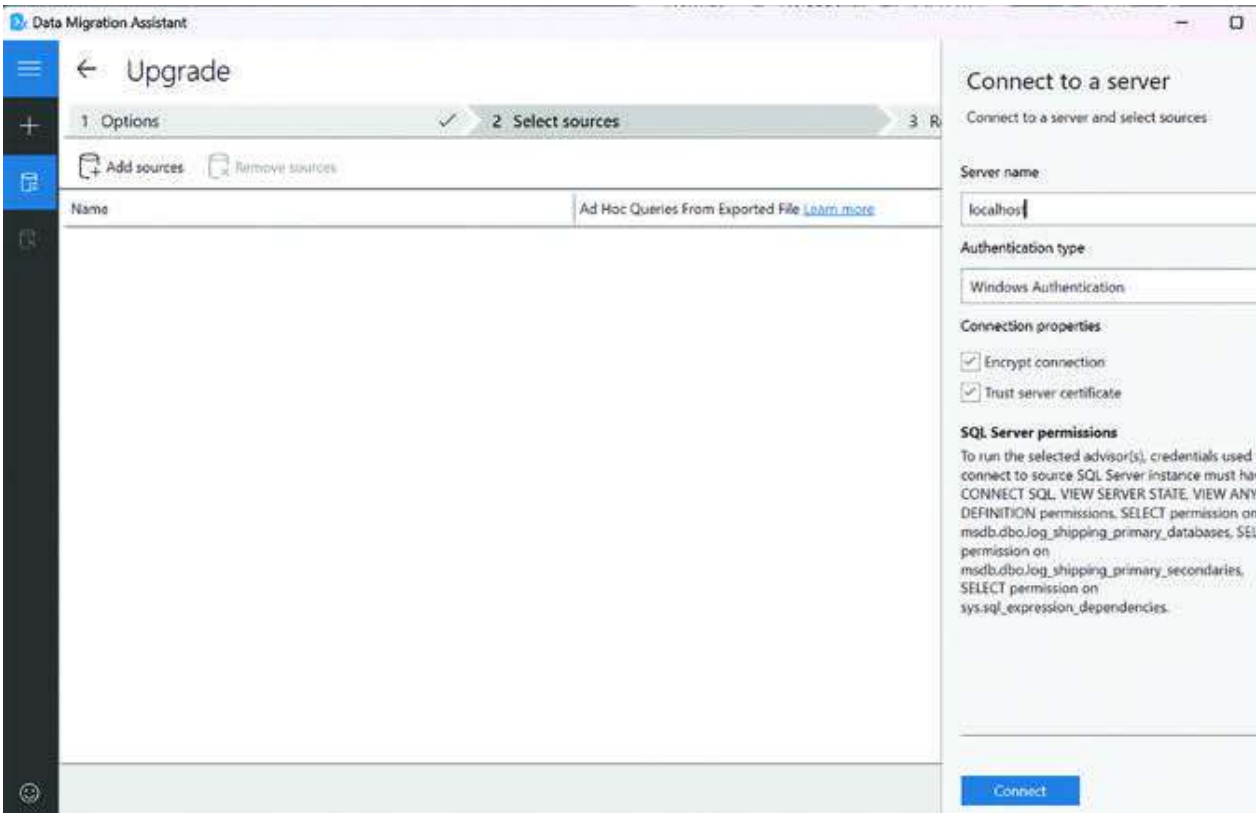


Figure 6.17: DMA Connect to a server blade

In the Add Source blade, select the source SQL Server instance and the database(s) to be assessed, and then click on

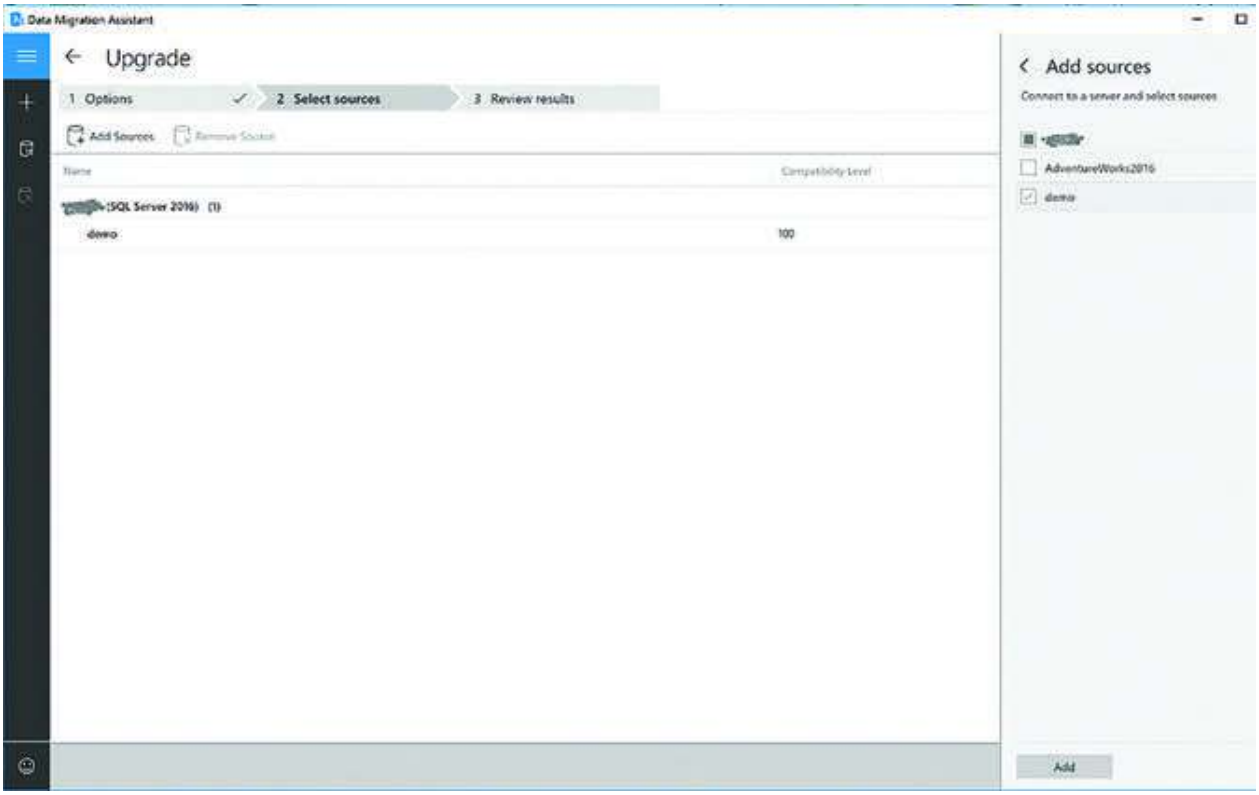


Figure 6.18: DMA Add Sources blade

Now, click Start

The assessment of the database(s) will take some time to deliver the results. The duration of the assessment depends on the number of databases added for assessment and the size of each database.

To view the reports, select the database, and expand Compatibility

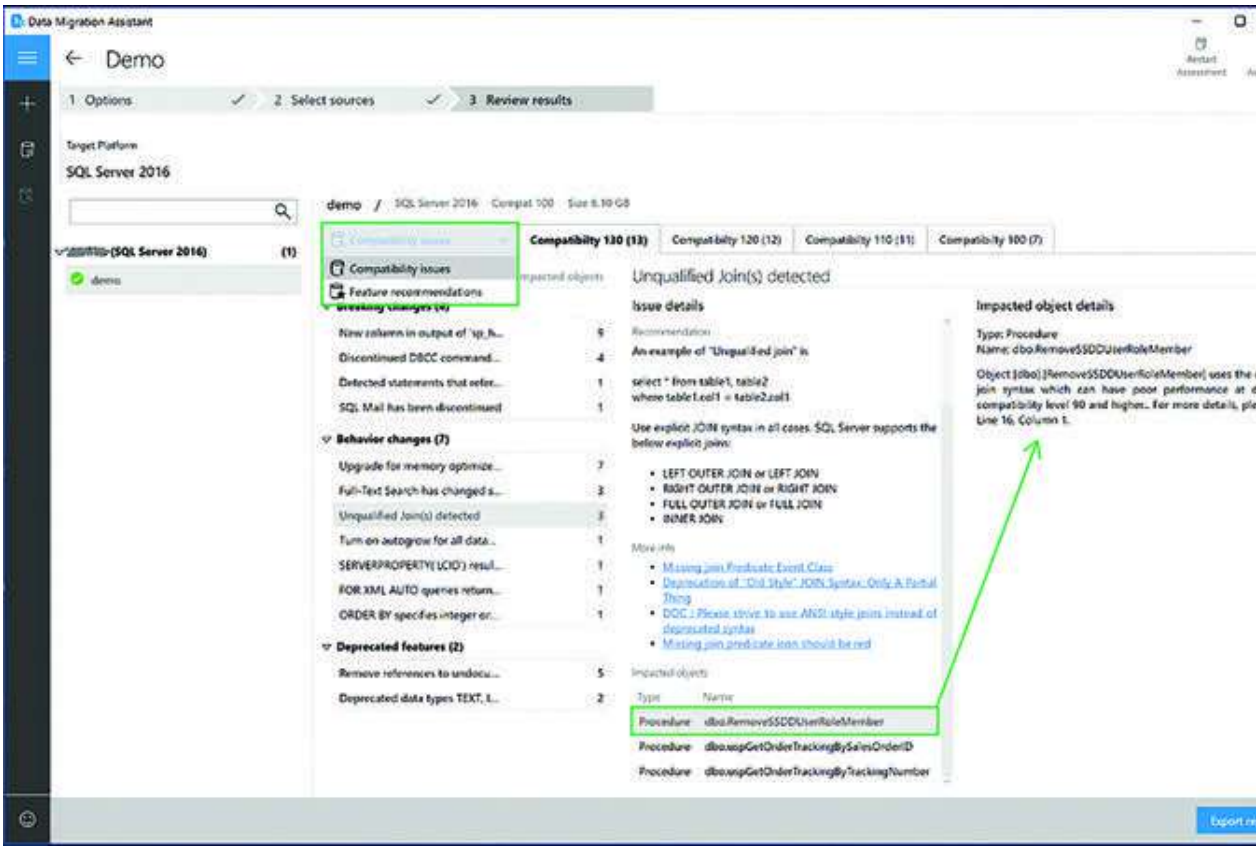


Figure 6.19: DMA Report blade

Once the issues are available for a review, analyze the affected object and its details for every issue identified under Breaking changes, Behavior and Deprecated

Feature recommendations cover various features such as Transparent Data Encryption (TDE), Dynamic Data Masking (DDM), In-Memory OLTP and Columnstore, and Always Encrypted (AE). After all database assessments are complete, Export report option would allow you to export the results to either a JSON or CSV file for analyzing the data.

Optional A/B Testing

This step is considered optional and not necessary to complete migration. But, during our engagements with multiple different customers who are sensitive to the performance of their queries for the mission-critical applications, we strongly recommended using this tool and evaluating the performance on the target SQL Server instance. This has helped them mitigate any potential issue that might arise after the migration is successfully completed.

Download the D, and then install it.

Run a trace capture on the source SQL Server database.

On the left navigation tree, select the camera icon and go to All

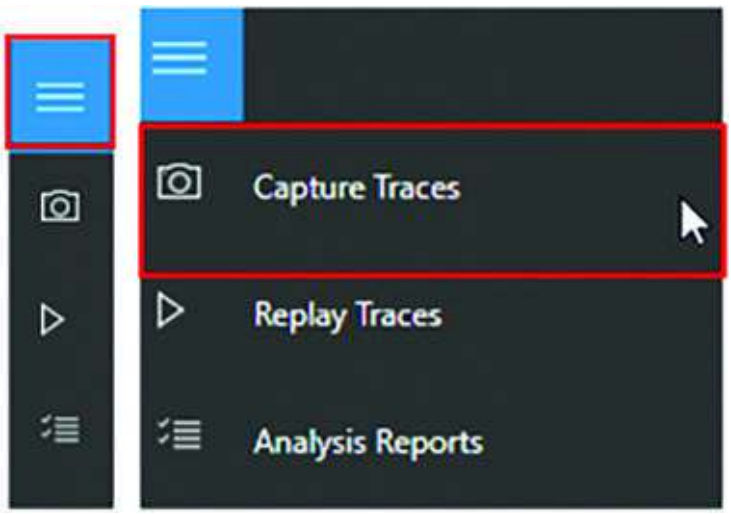


Figure 6.20: DEA Capture Traves blade

To start a new capture, select New

To configure the capture, specify the trace name, duration, target SQL Server instance name, database name, and the share location for storing the trace file

on the computer running SQL Server.

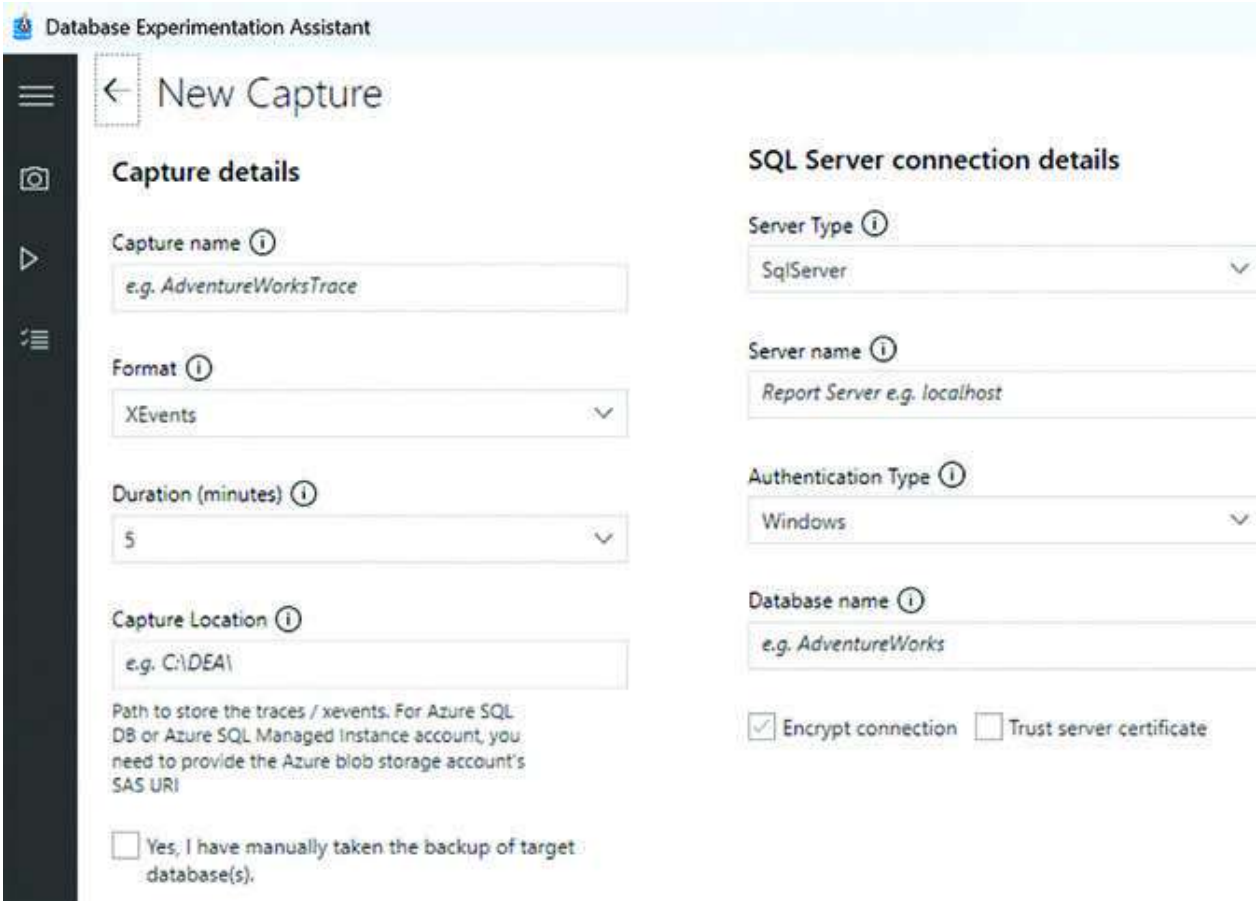


Figure 6.21: DEA Capture and Connection Details blade

Select Start to begin trace capture.

Run a trace

On the left navigation tree, select the play icon to go to All

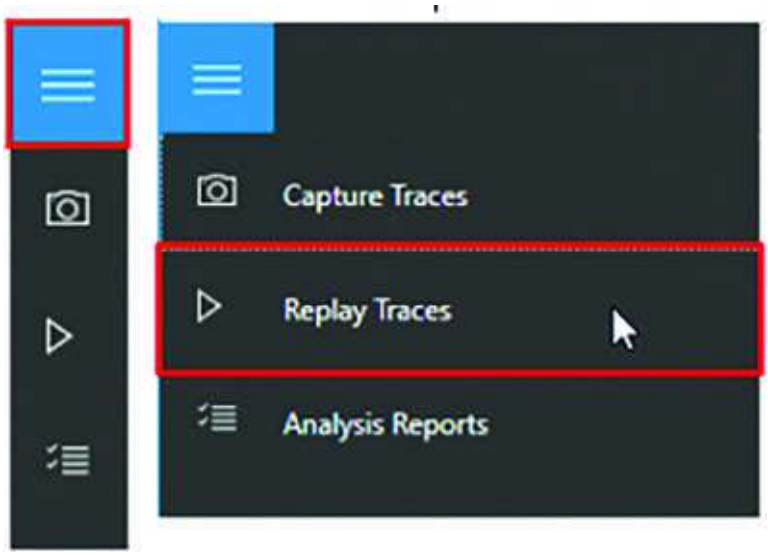


Figure 6.22: DEA Replay Traces blade

To start a new replay, select New

To configure the replay, specify the replay name, controller machine name, path to source trace file on controller, SQL Server instance name, and the path for storing the target trace file on the computer running SQL Server.

Select Start to begin replay of your capture.

Create a new Analysis Report.

On the left navigation tree, select the checklist icon to go to Analysis

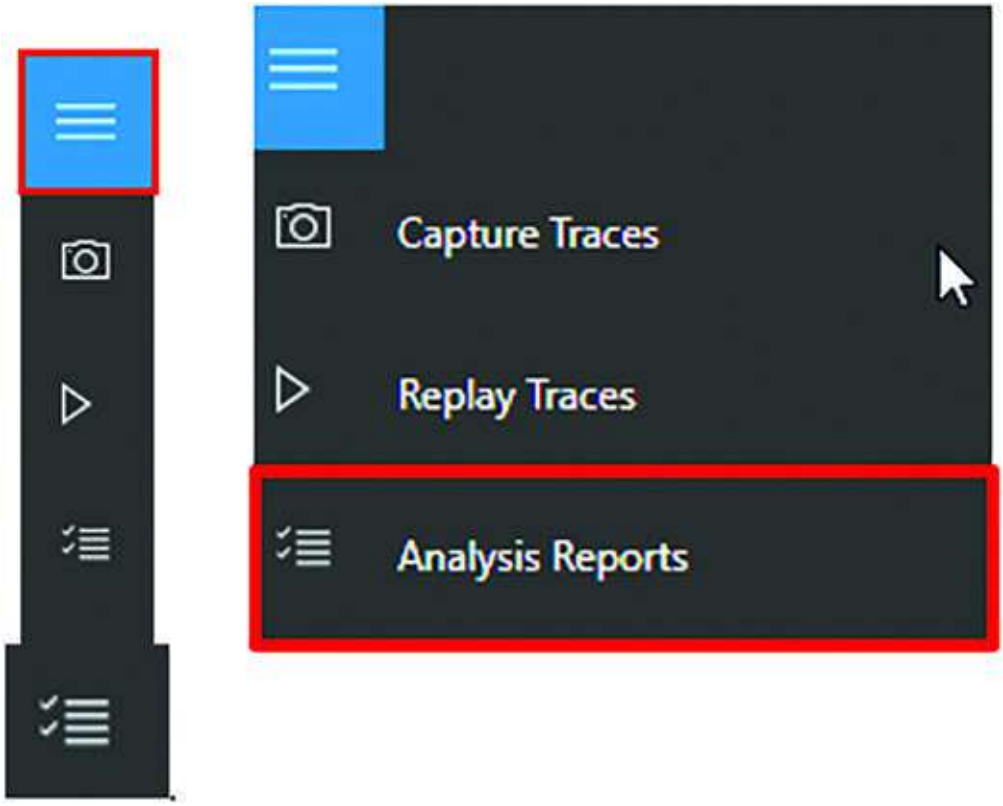


Figure 6.23: DEA Analysis Traces blade

Connect to the SQL Server on which you will store your report databases.

You will see the list of all reports in the server.

Select New

To configure the report, specify the report name, and specify paths to the traces for the source and target SQL Server instances.

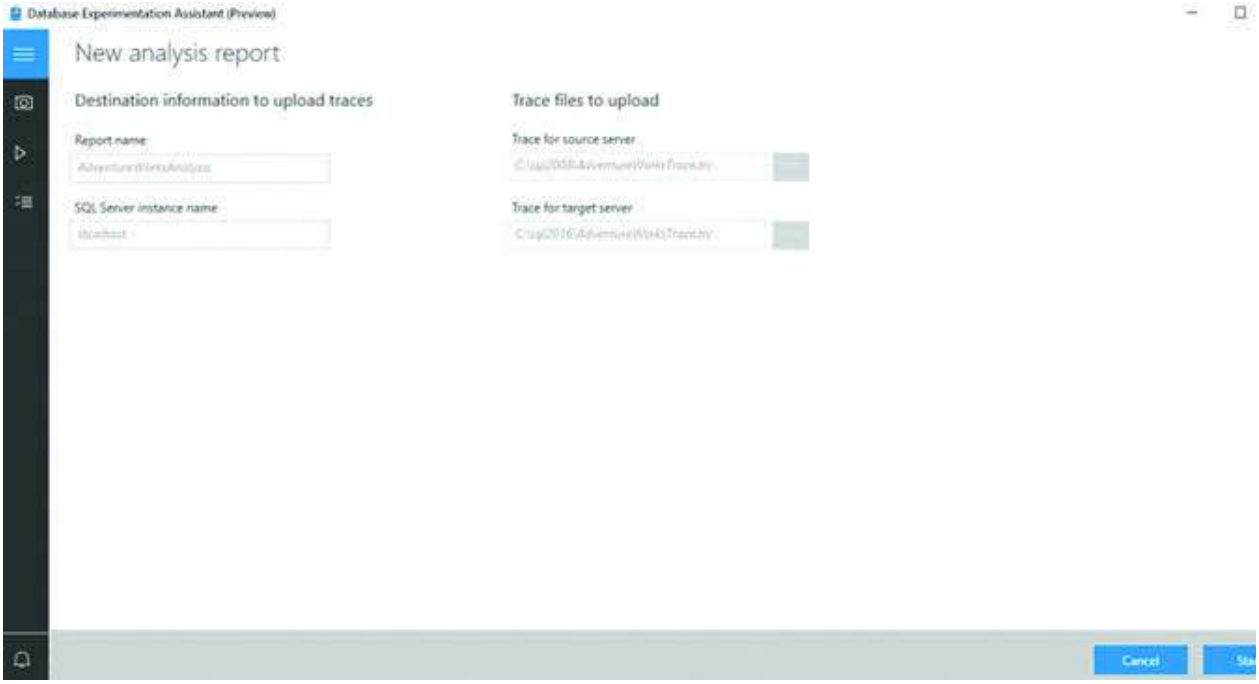


Figure 6.24: DEA Destination Information blade

Review of an analysis

On the first page of the report, the version and build information for the target servers on which the experiment was run displays.

Threshold allows you to adjust the sensitivity or tolerance of your A/B Test analysis.

default, the threshold is set to 5%; any performance improvement that is >= 5% is categorized as ‘Improved’. The drop-down selector allows you to evaluate the report using different performance thresholds.

Select the individual slices of the pie chart to view detailed metrics on performance. On the detail page for a performance change category, you will see a list of queries in that category.

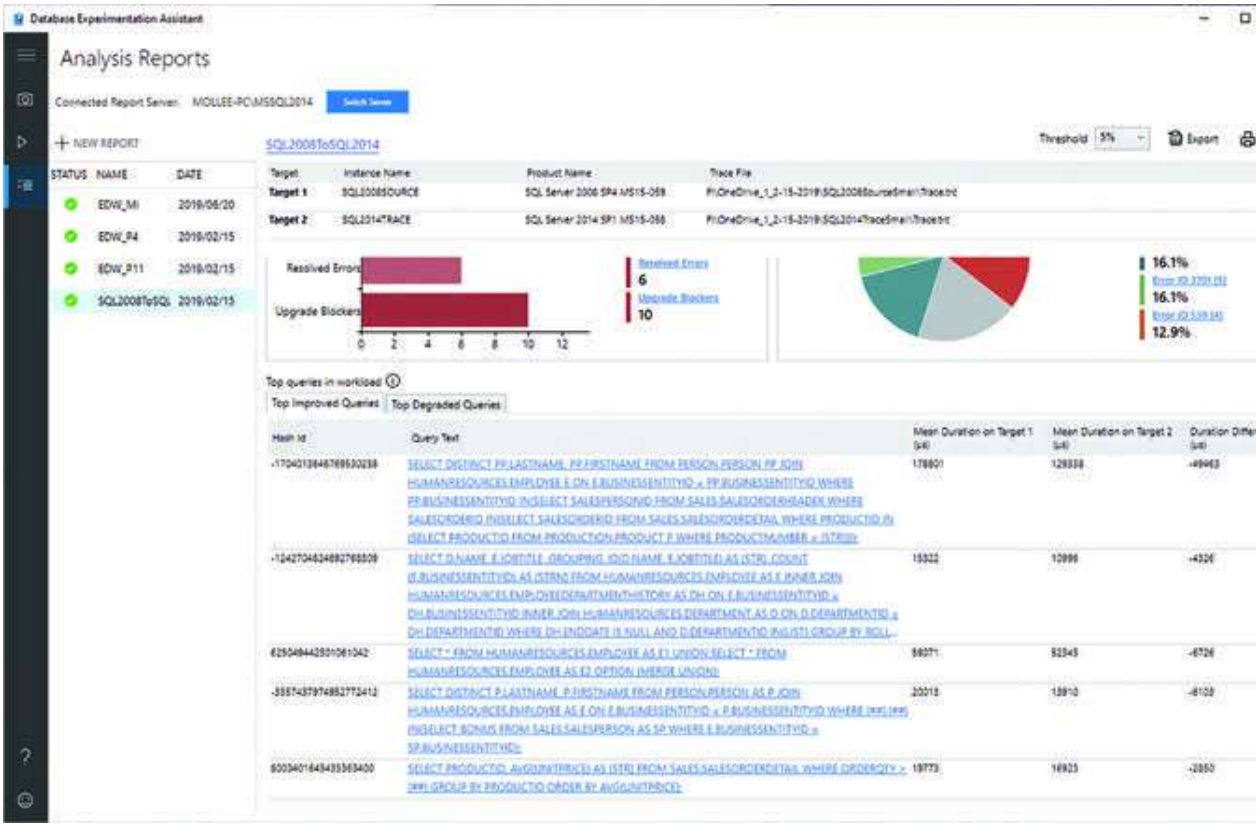


Figure 6.25: DEA Reports blade

Select an individual query to get performance summary statistics, error information, and query plan information.

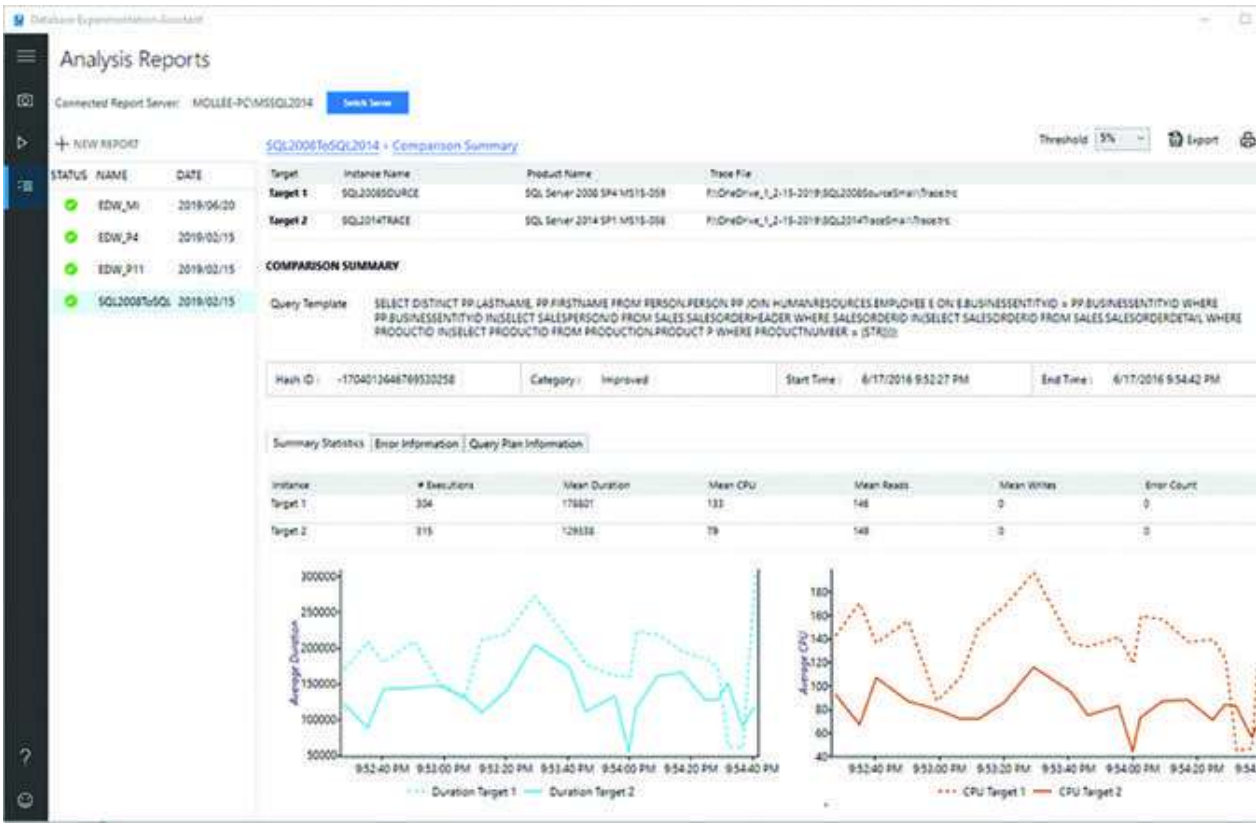


Figure 6.26: DEA Summary Statistics blade

Migration Phase

At this point in the migration journey, all the necessary prerequisites are in place and the tasks associated with the Pre-migration stage have been completed. Depending on which type of upgrade you choose, there are some important aspects to consider:

In-place Upgrade:

Preserve full database backup, differential, and transaction log backup for all databases, including system databases of the source SQL Server instance.

For TDE enabled databases, you must make sure to back up the master key and certificate that are used for TDE to a safe location. The master key and certificate are necessary to restore backups that were taken when TDE was enabled on the database to encrypt it.

These two steps are necessary in case the in-place upgrade fails in between due to any reason, then you can always rebuild the system and bring it to the original state by restoring the database backups.

The SQL Server Installation Wizard offers a unified feature tree for upgrading SQL Server components in-place to the latest version. While the steps for an in-place upgrade are not covered in this book, it is important to note that the previous version of SQL Server will be overwritten and will no longer be available on your computer.

Side-by-Side Upgrade:

In a side-by-side upgrade, it is imperative to migrate data and schema to the new environment.

Migrating Data and Schema

Well, this step is something wherein multiple strategies could be used to migrate schema and data together, schema only or data only. This depends on the size of the database, how much downtime organizations could tolerate, the complexity of the database, infrastructure requirements, and so on. But, above all, it depends on the choice of the tool. There are a wide variety of tools and techniques available both from Microsoft and third parties; however, we will discuss only those tools and techniques that are published by Microsoft or are available out of the box and are most relevant in migration activity.

Backup and Restore

This method involves taking a full database backup, differential backups (not necessary though), and all the transaction log backups from the source SQL Server and restoring them on the newly installed SQL Server 2022.

Simple and straightforward. Preserves the entire database, including schema, data, and objects.

This will incur a small downtime during the restore process or cutover, provided all the transaction log backups are preserved and are being restored. In the absence of transaction log backups, the downtime will be huge and will be incurred for the entire duration of the full database backup and restore. Another point to note is that both source and target SQL Server versions must be compatible.

Database Detach and Attach

In this method, detach the database from the source SQL Server, move or copy the database files to the target server, and then attach the database to SQL Server 2022.

Pros: Preserves the entire database.

Cons: Requires compatibility between source and target SQL Server versions. Involves manual steps. Downtime will be incurred right from the point of detachment operation until the database is attached. If the size of the database is huge, the copy file activity will take time and increase the downtime.

Data Export/Import (BCP, SSIS, or Import/Export Wizard)

This method exports data from the source SQL Server using tools such as Bulk Copy Program (BCP), SQL Server Integration Services (SSIS), or the SQL Server Import/Export Wizard and imports the data into the SQL Server 2022.

It allows for data transformation during migration. Suitable for complex data migration scenarios.

Cons: This requires additional effort for schema migration and handling data transformations. This method is not optimal for large databases.

Log Shipping

Log shipping involves shipping transaction logs from the source SQL Server to the target SQL Server and applying them to keep the databases synchronized. This method is somewhat like the backup and restore method we discussed previously, involving full database and transaction log backups, but is automated through out-of-the-box feature.

Near-real-time data replication with minimal downtime during cutover.

Easy setup, typically used for high-availability and disaster recovery scenarios. Failover requires manual effort.

Always On Availability Groups

The Always On availability groups feature is a high-availability and disaster recovery solution between two SQL Servers running on two different machines through Windows Failover Clustering.

Pros: Near-real-time data replication with minimal downtime during cutover. The failover functionality is quite easy, which requires minimal manual effort and can be a controlled, planned failover. This method is

one of the best for migrating large databases. Multiple databases can be replicated in parallel, saving time and effort.

Always On setup is a bit complicated as it requires target SQL Server to be a part of Windows Failover Clustering and requires many other pre-requisites before data could be replicated.

Data Migration Assistant

As we had discussed before, DMA is a great tool for assessing databases for any compatibility issues with SQL Server 2022. However, DMA also supports schema and data migration, allowing you to move both database schema objects (tables, views, stored procedures) and data to SQL Server 2022.

Pros: Easy to setup and use, multiple databases can be migrated in parallel.

Cons: Migrating very large databases with DMA can be complex and time-consuming, even with parallel data migration support.

Depending on your migration needs, you may need to take additional steps, such as migrating login accounts, updating connection strings, or reconfiguring application endpoints, irrespective of the choice of these tools and techniques.

It is to be noted that the aforementioned set of tools and techniques is not an exhaustive list to migrate data and schema to target SQL Server 2022.

However, these are the ones that are most widely used and proven to be the best.

Flying Off to Cloud

Azure SQL is a family of SQL database services that is fully managed, secure, and intelligent. Azure offers the widest range of deployment options for SQL from edge to cloud. You can rehost SQL workloads on SQL Server on Azure Virtual Machines, modernize existing applications with Azure SQL Managed Instance, and support modern cloud native applications with Azure SQL Database and Azure SQL Edge. Azure SQL is built upon the same SQL Server engine, so you can migrate applications easily and continue to use the tools, languages, and resources that you know the best. The services within Azure SQL support a variety of scenarios, as depicted in Figure

SQL Server on Azure Virtual Machines

Lift-and-shift your SQL workloads with ease and maintain with 100% SQL Server compatibility and operating system-level access. This flavor of Azure SQL allows full control and customization, including OS access.

Azure SQL Database

This managed service supports modern cloud native applications. The best option for developing native cloud applications is its ability to offer both high elasticity and flexibility. Developers can choose from different compute tiers, such as the Serverless tier for greater elasticity or the Hyperscale tier for highly scalable storage and compute resources.

Azure SQL Managed Instance

Modernize your existing SQL Server applications at scale with an intelligent, fully managed service. Azure SQL Managed Instance is the right PaaS target to modernize your existing SQL Server applications at scale, providing almost ninety-five percent of all SQL Server features (including instance-level features) while reducing the costs of server and database management.

SQL Server on Azure VMs and SQL Managed Instance are also now Azure Arc enabled, allowing you to run these services on the infrastructure of your choice when a hybrid approach is required.

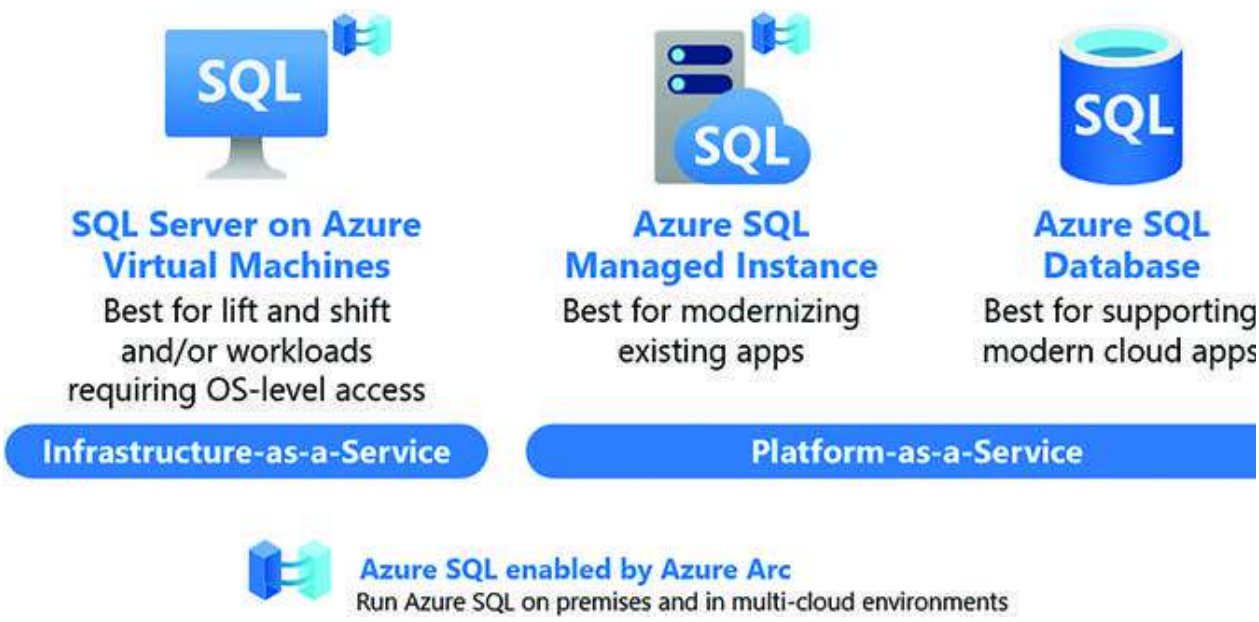


Figure 6.27: Family of SQL on Cloud

There are several benefits that organizations look forward to when they are in the process of deciding to move their SQL based workloads to cloud. This is a journey, and there are milestones that several organizations must meet in order to successfully choose the platform of their choice. Following are some of the benefits of each of these offerings available on the Azure platform.

Benefits of SQL Server on Azure Virtual Machines

SQL Server on Azure Virtual Machines saves money and simplifies management of security and high availability at no additional cost. Some of the top benefits organizations get after migrating to SQL Server on Azure Virtual Machines are:

Lower Through flexible pricing options, Azure hybrid benefits for customers with active Software Assurance, Azure Reservations, which can help save money by committing to one-year or three-year plans for multiple products, predictability of budget by paying upfront or monthly, free extended security support, and more.

Simple and familiar SQL Server: By incorporating existing skills, tools, and frameworks into the cloud seamlessly.

Powerful Performance: According to a study commissioned by Microsoft and conducted by GigaOm in January 2023, customers can achieve mission-critical performance for SQL Server on Azure Virtual Machines up to 57% faster and at up to 54% lower cost compared to AWS EC2, based on price-performance. This is with the Azure Hybrid Benefit and a three-year commitment.

Robust and simple HA/DR: SQL Server on Azure Virtual Machine provides end-to-end Availability Group deployment experience makes setting up HA/DR a breeze.

Built-in Security and Manageability: It ensures automatic security updates and simplified storage configuration management. Advanced threat protection, vulnerability assessment, data discovery and classification, Azure key vault integration, and Azure AAD authentication for SQL Server 2022 are some of the features that organizations look forward to.

Simple Migration with100% SQL Server Compatibility: It allows organizations to easily bring SQL Server to the cloud and enjoy versatile and powerful virtual machines.

Benefits of Azure SQL Database

Azure SQL Database helps accelerate innovation through leading performance, availability, and security. Integrated with the Microsoft Intelligent Data Platform, Azure SQL Database enables even higher savings. Some of the top benefits organizations get after migrating to Azure SQL Database are:

Simplified Development Experience allows building applications on popular platforms with support for the most widely used languages, efficiently meeting the needs of today's demanding apps with multimodal capabilities, and testing locally and deploying globally with a full-fidelity SQL Database emulator.

Near real-time insights with Azure Synapse Link for SQL lets minimum impact on operational workload, reduced complexity with no ETL jobs to manage, and near real-time insights into your operational data.

Fully managed and cost-effective: Azure SQL Database automatically scales compute based on workload demand with Azure SQL Database serverless, reduces costs and management complexity with Azure SQL Database elastic, pools and ensures optimal query performance with adaptive technologies.

Reduce your total cost of ownership: Azure Hybrid Benefit is a licensing benefit that helps to reduce significant costs of running the workloads in

the cloud. It works by letting you use your on-premises Software Assurance-enabled SQL Server licenses on Azure. You can further reduce your spend by pre-committing to fully managed Azure data services. Pay upfront or pay on a monthly basis at no additional cost with reservation pricing.

Rapid scalability and flexible performance: Azure SQL Database allows growth in storage as needed, backs up data almost instantaneously and restores it in minutes with Azure SQL Database Hyperscale, offers reduced latency with In-memory OLTP, and optimizes compute and storage for your specific requirements with flexible resourcing models.

Benefits of Azure SQL Managed Instance

Part of the Azure SQL service portfolio, Azure SQL Managed Instance is the intelligent, scalable, cloud database service that combines the broadest SQL Server engine compatibility with all the benefits of a fully managed platform as a service. While most of the aforementioned benefits of Azure SQL Database applies to Azure SQL Managed Instance as well, however, some of the additional benefits organizations get after migrating to Azure SQL Managed Instance are:

Always up to date with the latest SQL features: Azure SQL Managed Instance is based on the SQL Server engine and is always current with the latest SQL features and functionalities. You will never have to worry about updates, upgrades, or the end of support again.

Experience instance-level compatibility with SQL Server: Azure SQL Managed Instance provided SQL Server instance-level compatibility, and use the skills and experience you already have with familiar SQL tools and resources.

Fully Isolate and Secured Data: Azure SQL Managed Instance offers native virtual network support, ensuring that workloads remain fully contained within a secure virtual network. This allows you to enjoy the benefits of the public cloud while staying isolated from the public internet.

Operate more efficiently with a managed service: Azure SQL Managed Instance performs time-consuming tasks on your behalf, with features such as built-in high availability and disaster recovery, automated backup and point-in-time restore, and Azure Active Directory for identity management.

Database Migration

Database migration is the process of moving a database from one platform to another. The platform could exist in the on-premises environment, or it could be cloud. The migration process may involve moving the entire database or its schema and data. There could be the following scenarios that may involve database migration:

Platform Shift: This could be a movement of database from one RDBMS to another. For example, migrating from Oracle to Microsoft SQL Server.

Database Upgrade: This involves upgrading an older version of the database system to a new version of the same database system. For instance, moving from SQL Server 2016 to SQL Server 2022.

Cloud Modernization: This is primarily a shift from an on-premises database environment to a cloud database service, such as migrating from an on-premises SQL Server 2019 to an Azure SQL Database.

Infrastructure Relocation: In this scenario, the database is relocated from one physical data center to another as an effort of broader infrastructure changes.

Database Consolidation: This exercise involves consolidating multiple databases into a single database and requires moving data from multiple databases into a data warehouse for better analytics.

Specifically speaking about database migration tools for SQL Server migration scenarios, there are a few great tools available in the market that are published by Microsoft. Although there are a bunch of great third-party tools also available, we will discuss only the tools published by Microsoft. The discussion will emphasize the Azure Databases SQL Customer Success Engineering team, which is a part of the Azure SQL Engineering division at Microsoft. This team primarily engages with strategic customers to cater to their needs of modernization and migration of both homogeneous (SQL–SQL family) and heterogeneous (Non-SQL to SQL family or Non-SQL to Non-SQL) source and target pairs. This team has written a number of Intellectual (IPs) in the form of whitepapers, blogs, code artifacts, and tools to unblock customers in their migration efforts.

In the Migration Process Flow section, we talked about the discover, assess, and migration phases, which are critical for any database migration scenario. For the SQL Server instance upgrade scenario, we had discussed the MAP Toolkit for discovery, the Data Migration Assistant for database assessment, and other out-of-box tools for database migration. The techniques and tools are different for cloud migration scenarios, and in the next few sections we will talk about what tools to use for discovery, assessment, and data migration for SQL Server to Azure SQL IaaS and SQL based PaaS offerings, namely, Azure SQL Database and Azure SQL Managed Instance.

Discover Phase

Azure Migrate:

The Azure Migrate service is an excellent tool for discovering and assessing on-premises servers, infrastructure, apps, and data for migration to Azure at scale. Azure Migrate offers to migrate servers, databases, and applications all at once. This is a centralized hub for all the activities related to assessment and migration scenarios. From SQL Server perspective, Azure Migrate helps discover SQL Servers across datacenters and assess application dependencies. It by default understands the readiness of SQL Servers instance to get migrated to SQL Server Azure VM, Azure SQL database, and Azure SQL Managed Instance and gets recommendations, such as the optimal Azure SQL deployment option and the correct SKU that can fit the performance requirements for the workloads.

Azure Migrate is a widely used tool in the following scenarios:

Assessment and discovery of SQL Server data estate.

Get Azure SQL Database and Azure SQL Managed Instance deployment recommendations, target sizing, and monthly estimates.

Lift and shift the entire data estate to SQL Server on Azure VMs.

As mentioned earlier, Azure Migrate is a centralized hub that contains multiple different tools for various activities related to migration. Here are some tools that are critical for various phases of migration:

migration:
migration: migration: migration: migration: migration: migration: migration: migration: migration: migration: migration: migration:
migration: migration: migration: migration: migration: migration: migration: migration: migration: migration: migration: migration: migration: migration: migration: migration:
migration: migration:

Table 6.3: Azure Migrate Tools

People may argue how Azure Migrate and the MAP toolkit are different. While the Azure Migrate discovery and assessment tool provides assessment to help with migration readiness and evaluation of workloads for migration to Azure, the Microsoft Assessment and Planning (MAP) Toolkit helps with tasks such as migration planning for newer versions of Windows and Windows Server operating systems. For these scenarios, Microsoft recommends continuing to use the MAP Toolkit.

It is important to understand that the discovery and assessment tool uses a lightweight Azure Migrate appliance that needs to be deployed on-premises. The appliance runs on a VM, or a physical server can be installed easily using a downloadable template. The appliance has the capability to discover servers and continually sends metadata and performance data to Azure Migrate. There is no agent involved, which means nothing is installed on the discovered servers. After appliance discovery is complete, discovered servers can be grouped and assessments can run against each group.

Assessment Phase

The assessment phase for databases involves evaluating and measuring the readiness of SQL Servers for migration to Azure SQL. It includes discovering and assessing the SQL Server instance and databases, pinpointing workloads ready for migration, identifying compatibility issues, and assessing migration risks.

Azure Migrate:

Earlier, we discussed that Azure Migrate is a great tool during the discovery phase of migration; however, it also does help in the assessment of SQL Server databases. Before migration, it is a good practice to run an assessment on databases to identify migration blockers (if any). To perform an assessment, an Azure Migrate appliance is required for the applicable environment, whether it be VMware, Hyper-V, or a physical setup. This appliance identifies and discovers on-premises servers and sends metadata and performance data to Azure Migrate. These assessments are performance-based assessments, which means these assessments are based on performance data collected from target servers. Ideally, there must be a waiting period of at least a day after starting discovery before creating an assessment, which helps in providing time to collect performance data with greater confidence.

There are multiple steps and complex algorithms involved in collecting and calculating performance data from source SQL Server instances:

The appliance collects a real-time sample data point every 30 seconds.

Then the collected sample data point is aggregated over a 10-minute window. To create the sample data point, the appliance selects the peak values from all samples. It sends the max, mean and variance for each counter to Azure.

Azure Migrate keeps on storing all the 10-minute data points for the last month.

When an assessment is created, Azure Migrate identifies the appropriate data point to use for rightsizing. Identification is based on the percentile values for performance history and percentile utilization.

For example, if the performance history is one week and the percentile utilization is the 95th percentile, the assessment sorts the 10-minute sample points for the last week. It sorts them in ascending order and picks the 95th percentile value for rightsizing.

The 95th percentile value makes sure you ignore any outliers, which might be included if you picked the 99th percentile.

If you want to pick the peak usage for the period and do not want to miss any outliers, select the 99th percentile for percentile utilization.

This value is multiplied by the comfort factor to get the effective performance utilization data for these metrics that the appliance collects:

CPU utilization (%)

Memory utilization (%)

Read IO/s and Write IO/s (Data and Log files)

Read MB/s and Write MB/s (Throughput)

Latency of IO operations

Azure Migrate checks for feature compatibility with SQL Server on Azure VM, Azure SQL Database, and Azure SQL Managed Instance. The assessment looks for features that are currently in use (such as SQL Agent jobs, linked servers, and so on) and tables, views, triggers, stored procedures, and more to recognize compatibility issues. If it does not find compatibility issues, the instance would be marked as Ready for the target deployment type (SQL Server on Azure VM, Azure SQL Database, or Azure SQL Managed Instance). There are other statues that you might encounter based on deprecated or unsupported features or blockers to migration and more, such as Ready with warning details, Ready with and so on. If the assessment could not find source SQL Server configuration meeting the desired configuration and performance characteristics of any Azure based SQL deployment options, then it will be marked as Not

There is another piece of information that Azure Migrate would provide that is important for migration scenarios: SKU recommendation, wherein it evaluates a specific service tier and Azure SQL configuration (SKU size) that can meet or exceed the on-premises SQL instance performance.

In summary, Azure Migrate can be thought of as a one-stop solution for discovery and assessment of SQL Server instances for migrating to Azure SQL based offerings.

Azure SQL Migration extension for Azure Data Studio:

The Azure SQL Migration extension for Azure Data Studio is the newest tool built by Microsoft to help assess database requirements, get the right-sized SKU recommendations for Azure resources, and migrate SQL Server database to Azure. Earlier, there used to be many standalone tools such as DMA, SKU recommendations, and data migration tools, and it was cumbersome for customers to use these tools separately. Around a couple of years ago, a decision was made to simplify the entire migration experience for customers. It was understood that the first touch point for any customer is to assess the database, followed by identifying which SKU to select and then data migration. So, consciously, it was decided to develop a new tool, which will be released as a lightweight extension to the next-gen tool called Azure Data Studio. The following are the key benefits of Azure SQL Migration extension for Azure Data Studio:

End-to-end migration The extension gives an end-to-end experience for migration readiness assessment and SKU recommendation. The SKU recommendation is made by collecting performance data from source SQL Server instances.

SKU recommendation engine: This engine collects performance data from the on-premises SQL Server instances and then generates right-sized SKU recommendations based on the Azure SQL target you choose. This is similar to what we have discussed in the Azure Migrate section.

One of the important factors of using Azure SQL Migration extension for Azure Data Studio is its ability to do assessment at scale for scenarios when you have multiple servers that require Azure readiness assessment. The entire process can be automated using either Powershell based scripts module) or CLI extensions or Data Migration Assistant command line interface.

Assessment at scale using Powershell:

The assessment of multiple SQL Server instances is made possible using cmdlet of Az.DataMigration module.

Get-AzDataMigrationAssessment cmdlet expects the following parameters to execute:

Connection string of the SQL server you want to run assessment on. You can also provide multiple connection strings to run assessments on multiple SQL servers.

Output folder path where you want the assessment report to be stored. If this parameter is not provided, a default output folder depending on the OS platform is used.

This is a switch parameter that can be enabled to overwrite the existing report in that output folder. Currently, enabling this parameter is the only way to save the assessment report.

Following is the script to assess a single SQL Server:

```
Get-AzDataMigrationAssessment -ConnectionString “Data Source=LabServer.database.net;Initial Catalog=master;Integrated Security=True” -OutputFolder “C:\Output” -Overwrite
```

To do an assessment on multiple servers at scale, a config file is required, and this file can be passed to the Get-AzDataMigrationAssessment cmdlet.

The config file has the following structure:

```
{
“action”: “Assess”,
“outputFolder”: “C:\\Output”,
“overwrite”: “True”,
“sqlConnectionStrings”: [
“Data Source=LabServer1.database.net;Initial Catalog=master;Integrated Security=True;”,
“Data Source=LabServer2.database.net;Initial Catalog=master;Integrated Security=True;”
]
}
```

The config file can be passed to the cmdlet in the following way:

```
Get-AzDataMigrationAssessment -ConfigFilePath “C:\Users\user\document\config.json”
```

The following gives a simple sample of the Assessment report structure.


```
{
“Status”: “”,
“AssessmentId”: “”,
“Servers”: [
{
“ServerAssessments”: [],
“TargetReadinesses”: {
“AzureSqlDatabase”: {},
“AzureSqlManagedInstance”: {}
},
“Properties”: {},
“Errors”: [],

“Status”: “Completed”,
“Databases”: [
{
“DatabaseAssessments”: [
{
“ServerName”: “”,
“DatabaseName”: “”,
“DatabaseRestoreFails”: false,
“FeatureId”: “”,
“IssueCategory”: “Warning”,
“ImpactedObjects”: [],
“MoreInformation”: “”,
“RuleMetadata”: {},
“RuleScope”: “Database”,
“AppliesToMigrationTargetPlatform”: “AzureSqlManagedInstance”,
“Timestamp”: “”
}
```

],
“TargetReadinesses”: {
“AzureSqlDatabase”: {},
“AzureSqlManagedInstance”: {}
},
“Properties”: {},
“Errors”: [],
“Status”: “Completed”,
“FeatureDiscoveryTimeElapse”: “”
}
]

}
],
“Errors”: [],
“StartedOn”: “”,
“EndedOn”: “”
}

Few Important Properties in the json report:

Contains the server level assessment report.

Contains the list of databases that are ready for migration to SQL Managed Instance.

Contains the list of databases that are ready for migration to SQL Database.

Contains database level assessment report.

Stores the list of different issues that were found during the assessment of the database.

Tells to which category does the given issue belong to. It can take values like Warning and Error.

Tells whether the restore backup operation will fail or not. In many cases, databases having issues with IssueCategory “Error” can also be migrated to SQL targets, given the issue does not cause the restore to fail.

Contains additional information on the issue.

Contains the details whether the given database is ready for migration to different SQL targets such as Managed Instance and Database.

If the assessment fails due to any error, the error property stores the error details.

Migration Phase

The migration phase involves actively transferring data from the source to the destination. There are various tools that are there in public to carry out data migration activity.

Data Migration Tools

There are several tools available to let you migrate SQL Server databases from on-premises to Azure SQL IaaS and SQL PaaS based offerings. Microsoft provides a comprehensive guide on how to migrate a SQL Server instance to an Azure SQL Database. Here, let us discuss in brief the typical migration tools that have been developed and managed by Microsoft, and the Microsoft engineering team is committed to continuously enhancing these tools with new features and bug fixes. Some of these tools are used to migrate SQL Server to Azure SQL IaaS environments, Azure SQL Database, while others are used to migrate SQL Server to Azure SQL Managed Instance only.

Azure SQL Migration extension for Azure Data Studio

In the preceding section, we discussed the Azure SQL Migration extension for Azure Data Studio, which is a great tool for database assessment and SKU recommendation. This tool can be utilized for assessment at scale. However, this is also a very powerful tool for data migration at scale and can be automated. The fundamental benefits, apart from assessment and SKU recommendation-related benefits, of this tool are as follows:

This tool is powered by the Azure Database Migration Service, which orchestrates data movement activities to deliver a seamless migration experience.

This tool supports both online and offline data migration. Online migration refers to migration that requires minimal downtime, while

offline migration refers to migrations where downtime persists throughout the migration.

The data movement is based on both a backup and restore mechanism and a logical copy of data (row by row copy). So, this tool requires a self-hosted integration runtime configuration to access and copy the backup files and to allow copying records from an on-premises environment to Azure. This self-hosted runtime will act as a bridge between the on-premises environment and the Azure ecosystem and will efficiently copy the data to Azure.

This tool also provides secure user experience for migrating Transparent Data Encryption (TDE) databases and SQL/Windows logins to Azure SQL.

The tool works for all SQL targets on Azure, such as SQL Server on Azure VMs, Azure SQL Database, and Azure Managed Instance.

The following are the migration modes which can be operated by Azure SQL Migration extension:

extension: extension:
extension:
extension:
extension:

Table 6.4: Azure Migrate Extension Migration Modes

Here, let us discuss in brief the various migration scenarios for both online and offline modes. While we will not cover these scenarios in detail, we will cover some important aspects of these scenarios.

At a high level, the migration scenarios for SQL Server to SQL Server on Azure Virtual Machine and SQL Server to Azure SQL Managed Instance for both online and offline modes are similar.

Migration Scenario: SQL Server to SQL Server on Azure Virtual Machine and SQL Server to Azure SQL Managed Instance

Mode: Online

In the online migration mode, the source SQL Server database can be used by applications for read and write purposes, while database backups are continuously restored on the target SQL Server on Azure Virtual Machine. Application downtime is limited to the duration of the cutover at the end of migration.

Applicable on migrating from SQL Server 2016 and above.

This method uses Azure Database Migration Service (DMS) and the self-hosted integration runtime to access and migrate database backups.

This method uses SQL Server native backups, both full database and transaction log backups, to achieve minimal downtime.

It does not take database backups itself, or neither initiate any database backups. However, the service uses existing database backup files for migration, which you may already have as part of the disaster recovery plan, for migration.

You may keep database backup files in an SMB network share, in which case you must create an Azure storage account that allows the DMS service to upload the database backup files.

If database backup files are already copied in an Azure storage account, self-hosted integration runtime is not required during the migration process.

Azure Storage Account must be created in the same region as the Azure Database Migration Service instance is created.

It is recommended to use compressed backups to reduce the time and network bandwidth while migrating large backups.

Once the database is restored on the destination SQL Server instance on Azure VM, a downtime is required for applications that connect to the database, and the timing of the cutover needs to be carefully planned with business or application stakeholders.

Cutover Process:

Take application downtime and make sure all incoming transactions to the source database are stopped.

Point the application to the target database in SQL Server on Azure Virtual Machines by making configuration changes in the connection string.

Take a tail log backup of the source database in the backup location specified (with network share or Azure storage account).

At this time, the source database must be in read-only mode.

Ensure all database backups have the status Restored in the monitoring details page of Azure SQL Migration extension for Azure Data Studio.

Select Complete cutover in the monitoring details page.

While the cutover process is in progress, the migration status changes from In-Progress to

Once the cutover is complete, the migration status changes to

Migration Scenario: SQL Server to SQL Server on Azure Virtual Machine & SQL Server to Azure SQL Managed Instance

Mode: Offline

The offline mode for SQL Server to SQL Server on Azure Virtual Machine works in the same way as in Online mode. However, there are a couple of changes that must be considered:

In offline migration mode, applications are completely disconnected with the SQL Server database, and there must be no write activity happening on the database while database backup files are restored on the target instance of SQL Server to Azure Virtual Machines.

An automatic migration cutover is initiated by DMS after all database backups are restored on the instance of SQL Server on Azure Virtual Machines. At this time, the migration status changes from In-Progress to

Migration Scenario: SQL Server to Azure SQL Database

Mode: Offline

Migrating data from SQL Server to Azure SQL Database does not yet support online mode. Microsoft is actively working on supporting online migration mode, but this is going to take some time before it is released in public. Here are the few considerations before migrating a SQL Server database to an Azure SQL Database using the Azure SQL Migration extension for Azure Data Studio:

This method does not use a database backup and restore mechanism to migrate the database.

The offline migration utilizes Azure Data Factory (ADF) pipelines for data movement and thus abides by ADF limitations. A corresponding ADF service is created when a database migration service is also created.

The fundamental mechanism of migrating the data is called logical data movement, which is a row-by-row copying of data using ADF.

While the process uses ADF, a self-hosted integration runtime is required while setting up Azure Database Migration Service (DMS).

Since this is a logical data movement, database schema from source to target must be copied to the target database. This can be done by using the SQL Server dacpac extension or the SQL Database Projects extension in Azure Data Studio.

The machine where the self-hosted runtime is installed would compute for migration, which means this machine should be able to handle the CPU and memory load of the data copy process.

The speed of migration depends on the target Azure SQL Database SKU and the specifications of the self-hosted Integration Runtime host.

Azure Migrate

As discussed in the preceding sections, Azure Migrate is a comprehensive tool for end-to-end SQL Server migration scenarios. This includes discovery, assessment, and migration. From a migration perspective, the Azure Migrate hub includes the following two tools:

Data Migration Assistant (DMA)

Azure Database Migration Service (DMS)

While DMA is a great tool for migrating small migrations, both in terms of number and size of databases, it does not do a great job in migrating very large databases or large numbers of databases at scale, say of size greater than 10 TB. Also, DMA does not support migrating databases to Azure SQL Managed Instance, for which other tools such as the Azure SQL Migration extension for Azure Data Studio are a better option.

Azure DMS as a standalone service does not serve the purpose of efficient migration; however, as mentioned in the previous section, the Azure SQL Migration extension for Azure Data Studio uses Azure DMS in the background and is quite efficient in providing end-to-end migration experience.

Data Migration Assistant

While Data Migration Assistant (DMA) helps in assessment of the database with compatibility issues, it also recommends reliability and performance improvements for the target SQL Server or Azure SQL offerings. In terms of database migration, DMA helps in moving database schema, data, and uncontained objects from source to target SQL Server. As mentioned in the earlier section, DMA should not be used for large data migrations or database migration to Azure SQL Managed Instance.

Heterogeneous Migrations

In terms of database migration, we talk about homogeneous migration and heterogeneous migrations. In homogeneous migrations, a database engine is migrated to the same database engine or a different version. For example, SQL Server 2016 to SQL Server 2022 or SQL Server 2022 to Azure SQL Database. Whereas heterogeneous database migration is the process of migrating a database from one database engine to another. For example, migrating a database from Oracle 12c to Azure SQL Database would be considered a heterogeneous migration. Unlike homogeneous migrations, this process typically involves two steps:

Converting the source database schema and code to match that of the target database.

Migrating data from the source database to the target database, which sometimes involves data transformation to match the target schema.

It is imperative to say that heterogeneous migrations could become complex due to differences in database schema, data types, features and functionalities, and database code between the source and target databases. However, tools such as SQL Server Migration Assistant (SSMA) help simplify the migration process by handling data type conversions and providing automated schema conversion capabilities.

Currently, SSMA allows migration from Access, DB2, MySQL, Oracle, and SAP ASE to targets such as SQL Server 2012 and above, Azure SQL Database, Azure SQL Managed Instance, and Azure Synapse Analytics (for Oracle only). The SSMA for each of these sources can be separately downloaded from the Microsoft website. By far, in our experience, we have seen most of the heterogeneous migrations happening from Oracle and IBM DB2\IBM Mainframe to SQL based database engine.

Looking at the industry trends and migration patterns, there is huge potential in Oracle to SQL migrations. To tap early opportunity, Microsoft is coming out with several different tools and features to support customers for friction-free migration experiences. One such tool is Database Migration Assessment for Oracle As of now, this tool is in preview and may GA in the coming few months. The Database Migration Assessment for Oracle extension in Azure Data Studio helps assess an Oracle workload for Azure SQL and Azure Database for PostgreSQL targets. The extension identifies an appropriate sizing recommendation for Azure SQL or PostgreSQL target and suggests the complexity of the migration.

Heterogeneous Migration Assets

Following is a list of a few high-quality migration assets that were developed in support of a real-world migration project engagement by the Azure Databases SQL Customer Success Engineering.

Engineering.
Engineering. Engineering.
Engineering. Engineering.

Engineering. Engineering.
Engineering. Engineering.

Table 6.5: Heterogeneous Migration assets

Conclusion

Organizations face various drivers that compel them to modernize their data estate. These drivers include the need for improved performance, scalability, security, and cost efficiency. Depending on their specific business goals, organizations choose different migration patterns. Rehosting (Lift and Shift) involves moving existing applications and databases to Azure with minimal changes. It is a quick way to migrate but does not take full advantage of cloud-native features. Refactoring (Rearchitecting) requires modifying applications to fully leverage cloud services. This might involve rearchitecting code, using platform-as-a-service (PaaS) offerings, and optimizing for scalability. Lastly, Rebuilding (Greenfield Development) involves building new applications from scratch in Azure, taking advantage of cloud-native capabilities.

There are several options for upgrading their SQL Server instances to SQL Server 2022. These include in-place upgrades (minimal disruption), side-by-side upgrades (parallel environments), and rolling upgrades (gradual transition). Key tools for SQL Server upgrades include the MAP Toolkit (for discovery and assessment), Data Migration Assistant (DMA, for compatibility assessment and migration), and Database Experimentation Assistant (for evaluating target SQL Server versions). During migration, organizations can choose from various techniques such as backup and restore, database detach/attach, data export/import (BCP, SSIS, or Import/Export Wizard), Log Shipping, and Always On Availability Groups. Azure offers different services for SQL workloads, including SQL Server on Azure Virtual Machines (IaaS), Azure SQL Database (PaaS),

and Azure SQL Managed Instance (PaaS). Each service provides unique benefits. Additionally, the data migration process involves moving the entire database or its schema and data separately. Tools such as Azure Migrate help discover and assess on-premises resources for migration, while the Azure SQL Migration extension in Azure Data Studio assists in migrating SQL Server databases to Azure. Migration scenarios include moving from SQL Server to Azure SQL Virtual Machine, Azure SQL Database, or Azure SQL Managed Instance, with options for online or offline modes. Finally, heterogeneous migrations allow databases from one engine (example, Oracle, MySQL) to be migrated to SQL Server using appropriate tools.

In the next chapter, we will discuss how to achieve SQL brilliance by adhering to best practices that optimize query performance, enhance code readability, and mitigate risks such as SQL injection attacks. Whether you are a seasoned database developer or a novice SQL enthusiast, adopting these practices can significantly streamline your workflow and elevate the quality of your code.

References

https://www.microsoft.com/download/details.aspx?id=103130

https://www.microsoft.com/download/details.aspx?id=103121

https://www.microsoft.com/download/details.aspx?id=103120

https://www.microsoft.com/download/details.aspx?id=103016

Microsoft Azure innovation powers leading price-performance for SQL Server | Azure Blog | Microsoft Azure

---

native applications are built from the ground up, optimized for cloud scale and performance. Some of them are based on microservices architectures, use managed services, and take advantage of CICD to achieve faster time to market.

Databases SQL Customer Success Engineering, nicknamed as SQL Ninja Engineering, has published some of their IPs in public here:

Some of their IPs are still internal, for which Ninja must be engaged through your Microsoft account team.

CHAPTER 7

Achieving SQL Brilliance Through Best Practices

Introduction

Establishing and maintaining an optimal SQL Server environment demands a nuanced approach, encompassing various facets from installation to ongoing security and performance considerations. The foundational step, often referred to as Day 1, involves the installation and configuration of SQL Server. This stage is dynamic, adapting to diverse deployment scenarios such as on-premises, Azure, Linux, or containerized environments. Successful configurations hinge on factors such as platform diversity, ensuring compatibility between SQL Server versions, operating systems, and hardware. Adequate resource planning, including CPU, memory, and storage allocation based on workload requirements, lays the groundwork for a stable environment.

Security is an integral aspect of SQL Server management, requiring meticulous attention to best practices. The landscape of security assessments and vulnerability management tools has evolved in tandem with the dynamic nature of cyber threats. Regular vulnerability scans, compliance checks, and adherence to industry best practices contribute to a proactive security stance, mitigating potential risks.

As SQL Server increasingly finds its home in Azure, performance optimization strategies must adapt to the cloud environment. Dynamic resource scaling becomes paramount, leveraging Azure SQL Database's elastic pools or virtual machine scaling to adjust resources based on workload demands. Efficient query optimization, facilitated by tools like

SQL Server Query Store or Query Performance Insight, enhances database performance and reduces resource consumption. Strategic indexing practices further contribute to streamlined query execution. The integration with Azure Monitor provides real-time visibility into performance metrics, enabling proactive issue resolution and ensuring the system operates optimally.

In this intricate dance of security and performance, SQL Server administrators must also consider the ever-evolving threat landscape. Azure's Threat Detection features provide an additional layer of defense, enabling the system to proactively identify and respond to potential security threats. By embracing threat intelligence and leveraging behavioral analysis, administrators can stay ahead of emerging risks.

Structure

In this chapter, we will cover the following topics:

Setting Up SQL Server Instance Configuration

Practices for SQL Server Database Configuration

Efficiently Making SQL Server Secured

Query Performance Optimization

Managing Resource Utilization Efficiently

Proactive Performance Monitoring

Day 1: SQL Server Configuration

As we embark on configuring the SQL Server environment, Day 1 sets the stage for establishing critical settings, security measures, and performance optimizations.

Instance Configuration

The foundational step on Day 1 following the installation of SQL Server 2022 marks a critical phase in the configuration process. With the installation as the cornerstone, attention shifts to tailoring the environment to meet specific needs and align with optimal performance standards. This involves navigating diverse deployment scenarios, including on-premises setups, Azure cloud, Linux environments, or containerized platforms. Thorough consideration is given to factors such as platform compatibility, version alignment, and high availability configurations. Adequate resource planning, encompassing CPU, memory, and storage allocations tailored to the anticipated workload, lays the groundwork for a resilient system. This foundational configuration ensures that SQL Server 2022 is not merely installed but fine-tuned to deliver optimal performance, setting the stage for a robust and adaptable database environment.

Following are some of the general considerations and best practices for improving the performance of the SQL Server instance.

Max degree of parallelism:

The “max degree of parallelism” (MAXDOP) Server Configuration Option in SQL Server determines the maximum number of threads that can be used for the execution of a single query. Configuring MAXDOP appropriately is crucial for optimizing performance. Analyze your workload to understand the nature of queries and parallelism

requirements. Some queries benefit from parallel execution, while others may not. While setting MAXDOP to 0 allows SQL Server to utilize all available processors (up to 64), it’s not recommended in almost all scenarios. For most systems, setting MAXDOP at 8 is often appropriate. This allows SQL Server to determine the degree of parallelism dynamically based on the workload and system resources.

Starting with SQL Server 2016, during service startup if the Database Engine detects more than eight physical cores per NUMA node or socket at startup, soft-NUMA nodes are created automatically by default. The Database Engine places logical processors from the same physical core into different soft-NUMA nodes. The following table is a representation of recommendations at keeping all the worker threads of a parallel query within the same soft-NUMA node. This will improve the performance of the queries and distribution of worker threads across the NUMA nodes for the workload. Microsoft has published guidelines for configuring max degree of parallelism based on certain server configurations. The following table summarizes these guidelines:

guidelines:
guidelines: guidelines: guidelines: guidelines: guidelines: guidelines: guidelines: guidelines: guidelines:
guidelines: guidelines: guidelines: guidelines:
guidelines: guidelines: guidelines: guidelines: guidelines: guidelines: guidelines: guidelines: guidelines: guidelines: guidelines: guidelines:

guidelines: guidelines: guidelines: guidelines: guidelines: guidelines: guidelines: guidelines: guidelines: guidelines: guidelines: guidelines: guidelines: guidelines: guidelines: guidelines: guidelines: guidelines:

Table 7.1: Degree of Parallelism Guidelines

While setting MAXDOP to 1 can be beneficial for certain workloads, it is generally not recommended to set it globally at the server level. This could limit the performance of parallelizable queries. Utilize query hints to set MAXDOP at the query level. This allows you to override the server-level setting for specific queries, providing more granular control. Implement changes to MAXDOP in a controlled testing environment first. Monitor performance metrics and query execution times to ensure that changes have the desired effect without causing negative impacts.

Optimize for ad hoc workloads:

The “optimize for ad hoc workloads” option in SQL Server 2022 is used to reduce the memory usage of single use ad hoc batches and associated plans. When enabled, it stores a compiled plan stub, which has a smaller memory footprint, on the first execution of an ad hoc batch. Plan stubs have lower memory usage compared to the size of the full compiled plan. A compiled plan stub is stored in the cache during the first compilation of the batch. In the subsequent compilation or execution of the batch, the compiled plan stub will be removed and replaced with a full compiled plan.

Maximum server memory:

Before configuring max server memory, have a clear understanding of the total physical memory available on the server. This information is essential for making informed decisions about memory allocation.

Reserve a portion of the total system memory for the operating system and other system-level processes which ensures that the OS has sufficient resources to operate efficiently.

Consider the other applications and services running on the server as they would have the memory demands too. Finally, allocate an appropriate share of memory to SQL Server, keeping in mind the needs of the entire system. Ideally, maximum server memory must be set to approximately 80-90 percent of total physical memory. Please avoid setting max server memory to an extremely high value, as it may lead to resource contention with the operating system. Instead, set a realistic limit that allows SQL Server to perform optimally without starving the OS. If the server hosts multiple SQL Server instances, then carefully allocate memory for each instance to prevent contention. The sum of the max server memory settings across all instances should not exceed the physical memory available.

Lock pages in memory:

Windows-based applications can use Windows Address Windowing Extensions (AWE) APIs to allocate and map physical memory within the process address space.

The LPIM (Lock Pages in Memory) Windows policy controls which accounts can access this API, ensuring that data remains in physical memory and preventing it from being paged to virtual memory on disk. Memory allocated using AWE remains locked until the application explicitly releases it or terminates. In the context of 64-bit SQL Server, using AWE APIs for memory management is often referred to as ‘locked pages.’ Enabling this option can help maintain server responsiveness

during memory paging to disk. The ‘Lock pages in memory’ option is available in SQL Server Standard edition and higher, as long as the account running sqlservr.exe has been granted the Windows Lock pages in memory (LPIM) user right.

Database Configuration

Configuring SQL Server databases is a critical step in ensuring optimal performance, security, and scalability. Let's delve into the key aspects of setting up and fine-tuning these databases.

Database and Log Files:

The placements and layout of a database and log file play a crucial role in determining the performance of the queries. Distribute database files across multiple disks to balance I/O operations and enhance performance. Place data and log files on separate disks to prevent simultaneous read and write operations on the same disk. Enable Instant File Initialization to speed up the process of allocating space for database files. Set appropriate autogrowth settings for database files to avoid frequent and unnecessary growth operations. Prefer fixed-size autogrowth over a percentage to prevent excessive fragmentation.

Backup and Restore Strategies:

Implement regular database backup and restore strategies to ensure data recoverability and adherence to RTO and RPO needs of the business. Some of the best practices for SQL Server backup and restore strategies are:

Regular Schedule full, differential, and transaction log backups to minimize data loss. Full backups should occur weekly, differential

backups daily, and transaction log backups frequently (for instance, every 15 minutes).

Backup Regularly test backups by restoring them to ensure data integrity and verify that the restore process works as expected.

Offsite Storage: Store backups securely offsite or use cloud storage to protect against physical disasters.

Backup Encrypt backups to safeguard sensitive data, especially when storing them offsite or in the cloud.

Compression: Utilize backup compression to reduce storage requirements and enhance backup performance.

Automated Alerts: Configure alerts for backup job failures to ensure timely resolution.

Retention Implement policies to automatically delete old backups, freeing up space and reducing clutter.

Point-in-Time Ensure frequent transaction log backups to support point-in-time recovery if needed.

Database Compatibility Level:

When Microsoft releases a new version of SQL Server, they typically introduce the latest features exclusively for databases running at the newest compatibility levels. For example, SQL Server 2022's Parameter Sensitive Plan Optimization (PSPO) is only available in databases with the SQL Server 2022 compatibility level. If you're migrating a database from an older SQL Server version to SQL Server 2022 and want consistent behavior, it is recommended to maintain the current compatibility level. For instance, if your database is currently at the SQL Server 2016 compatibility level, you can move it to a 2022 server while keeping the compatibility level at 2016. Users should notice minimal differences. However, it is important to understand that certain changes within SQL Server itself, regardless of compatibility level, may affect behavior. If Microsoft deprecates and removes a feature, it won't be available even in older compatibility levels.

If there is a particular feature you require that is only available in specific compatibility levels, then adjusting the compatibility level is necessary. However, if you're satisfied with the current performance, it's best to avoid changing it. Why? Altering the compatibility level can sometimes degrade performance rather than improve it. While there are cases where it significantly enhances performance, any change introduces an element of risk. Therefore, modifying the compatibility level when you're already content can be precarious.

When migrating from SQL Server 2019 to 2022, adjusting a database's compatibility level can impact performance in various ways:

Cardinality Estimation This feature can alter the estimated number of rows for queries.

It may improve or worsen query plan accuracy based on actual execution feedback.

Degrees of Parallelism Reduces overhead related to poor parallelism.

Note that most parallelism is beneficial, but this feature helps optimize it.

Parameter Sensitive Plan Allows caching of multiple query plans for the same query. Enhances plan reuse and performance.

Remember that each SQL Server version enables different features under newer compatibility levels. Before changing the compatibility level, review the added features specific to the new level. This serves as a valuable starting point for decision-making.

Index Maintenance:

Periodically evaluate the fragmentation level of indexes using tools like Rebuild heavily fragmented indexes (>30% fragmentation) and reorganize moderately fragmented indexes. Implement a regular schedule for index maintenance tasks to ensure optimal performance. Consider running maintenance tasks during off-peak hours to minimize the impact on production. Leverage intelligent maintenance plans or scripts that dynamically analyze fragmentation levels and selectively perform actions. Utilize algorithms that consider the size of the table, the degree of fragmentation, and the available maintenance window. Take advantage of the online index rebuilding feature in Enterprise Edition to minimize downtime during maintenance. Be aware that online index rebuilds might consume additional system resources. Adjust the fill factor based on the

level of fragmentation and the rate of data modifications. A lower fill factor can reduce fragmentation but increases storage requirements.

For partitioned tables, focus on maintaining individual partitions rather than the entire table. Rebuild or reorganize only the partitions affected by data modifications. Evaluate the necessity of each index and avoid over-indexing, as it can impact write performance and increase storage requirements. Drop redundant or unused indexes to streamline maintenance and improve overall system efficiency. Use scripts or third-party tools to automate the index maintenance process. Automating tasks ensures consistency and reduces the likelihood of manual errors.

For small tables, carefully consider the need for indexes, as the overhead of maintaining them might outweigh the benefits. Clustered indexes are particularly impactful on small tables, so assess their value judiciously. Regularly monitor the impact of index maintenance on system resources, such as CPU, memory, and I/O. Adjust maintenance plans accordingly to minimize disruption to other critical operations.

Partitioning:

Implement partitioning for large tables to improve manageability and query performance. Align partitioning with the application’s query patterns to achieve optimal results. Select a partitioning key that aligns with your query patterns and facilitates efficient data retrieval. Use a column that is frequently used in WHERE clauses and has a high cardinality. For time-series data, consider using a date column as the partition key. Numeric columns that exhibit a range of values are also good candidates. Align partitions with filegroups to distribute data and I/O operations evenly across multiple disks. Assign each filegroup to a

separate physical disk or storage device for better performance. If possible, use an incrementing key (such as an identity column) as part of the partitioning key, this helps in efficiently adding new rows to the latest partition. Keep partition-level statistics up-to-date to ensure the query optimizer makes informed decisions. Use the UPDATE STATISTICS command with the FULLSCAN option. Implement data compression on partitions to reduce storage requirements and enhance I/O performance. Choose the appropriate compression type based on your workload (ROW or PAGE compression).

Monitor the health and performance of partitions regularly. Schedule maintenance tasks to rebuild or reorganize indexes on partitions based on fragmentation levels. Use partition switching for efficient movement of data in and out of partitions. This is particularly useful for data archiving or purging. The optimal number of partitions depends on the size of your dataset and the characteristics of your queries. Experiment with different partitioning strategies and measure their impact on query performance. Leverage partitioned views to simplify querying and reporting across partitioned tables. The view can provide a unified perspective on data while the underlying tables are partitioned. Align indexes with the partitioning strategy to maximize their effectiveness. Partitioned tables often benefit from local indexes on each partition. Use filtered indexes that align with the partition key for queries that only access specific partitions. This can significantly improve query performance. As your data grows and query patterns change, regularly review and adjust your partitioning strategy.

Audit and Monitoring:

Leverage built-in reports in SSMS for quick insights into performance, index usage, and resource utilization. Utilize reports to identify performance bottlenecks and areas for improvement. Establish performance baselines for your SQL Server instances. Regularly compare current performance metrics to established baselines to identify deviations and potential issues.

Define and set up alerts for critical events, such as high CPU usage, low disk space, or excessive blocking. Utilize SQL Server Agent to manage and trigger alerts. Regularly review query execution plans to identify inefficient queries. Use tools like SQL Server Management Studio (SSMS) or third-party monitoring solutions. Leverage DMVs to obtain real-time insights into SQL Server performance. Query DMVs to identify long-running queries, blocked processes, and resource utilization. Track and analyze wait statistics to identify performance bottlenecks. Address the most frequent and impactful wait types affecting system performance. Implement a scheduled task to update statistics on tables and indexes. This ensures that the query optimizer makes accurate decisions based on up-to-date information.

Schedule SQL Server Agent jobs for routine maintenance tasks, such as backups, index maintenance, and database integrity checks. Monitor the status of these jobs to ensure they run successfully. Enable the capture of deadlock information using trace flags or Extended Events.

Analyze deadlock graphs to identify and resolve conflicts. Regularly review SQL Server error logs for warnings, errors, and informational messages.

Security Best Practices

Securing SQL Server 2022 is critical to safeguard sensitive data and protect against potential security threats. A layered security approach offers a defense-in-depth strategy by employing various security capabilities across different security levels. SQL Server 2016 introduced security features that have been further enhanced in subsequent releases, bolstering protection against security threats and ensuring well-secured database applications.

Additionally, Azure adheres to industry regulations and standards, allowing you to create compliant solutions when running SQL Server in a virtual machine.

Column Level Security

Organizations often need to enhance data protection at the column level, particularly when dealing with sensitive information in SQL Server databases. These sensitive columns might include personal data like social security numbers, mobile phone numbers, names, financial account details, and other confidential information. The techniques and features discussed in this context improve column-level protection with minimal impact and without necessitating significant modifications to application code.

Consider implementing Always Encrypted to secure data both at rest and during transmission. Encrypted data remains decrypted only at the application client level through client libraries. When choosing between randomized and deterministic encryption, prioritize randomized encryption where feasible. Additionally, leveraging Always Encrypted with secure enclaves can enhance performance for comparison operations such as BETWEEN, IN, LIKE, and Joins in scenarios involving randomized encryption.

Consider employing Dynamic Data Masking (DDM) to conceal data at the column level when Always Encrypted is not feasible. Note that DDM and Always Encrypted are not compatible. Whenever possible, prioritize using Always Encrypted over dynamic data masking.

You can also assign permissions at the column level for a table, view, or table-valued function. Keep in mind the following points:

Only SELECT, REFERENCES, and UPDATE permissions can be granted on a column.

A table-level DENY does not override a column-level

Row Level Protection

Row-Level Security (RLS) allows you to control access to rows in a database table based on user execution context. With RLS, users can only view records relevant to them, providing ‘record-level’ security without major application modifications.

The underlying business logic resides in table-valued functions governed by a security policy that enables or disables RLS functionality. This policy also manages FILTER and BLOCK predicates associated with the tables where RLS operates. To restrict returned records for specific users, leverage Row-Level Security (RLS). Additionally, consider using SESSION_CONTEXT (T-SQL) for middle-tier application users who share the same SQL Server user account.

Note: Use Row-Level Security (RLS) together with either Always Encrypted or Dynamic Data Masking (DDM) to maximize the security posture of your organization.

File Level Encryption

Transparent Data Encryption (TDE) ensures data security at the file level by applying encryption-at-rest to database files. TDE prevents unauthorized access to these files, including database backups and tempdb files, unless the appropriate certificates are used to decrypt them. Without TDE, attackers could potentially restore or attach a database from physical media (such as drives or backup tapes) to access its contents. TDE seamlessly integrates with other SQL Server security features. It offers real-time I/O encryption and decryption for data and log files. TDE relies on a database encryption key (DEK) stored within the user database, which can also be protected using a certificate secured by the master key in the master database. Consider using TDE to protect data at rest, backups, and tempdb.

Identities and Authentication

SQL Server offers two authentication modes: Windows authentication mode and ‘SQL Server and Windows Authentication mode’ (mixed mode).

Logins are distinct from database users. Begin by mapping logins or Windows groups to database users or roles separately. Then, grant permissions to users, server roles, and/or database roles to access database objects.

SQL Server supports the following login types:

Local Windows User Account or Active Directory Domain SQL Server utilizes Windows for authenticating user accounts.

Windows Assigning access to a Windows group grants permissions to all Windows user logins within that group. Conversely, removing a user from the group results in the withdrawal of their associated rights. Group membership is the recommended method.

SQL Server Within the master database, SQL Server stores both the username and a hashed password

Contained database users authenticate SQL Server connections at the database level. A contained database is isolated from other databases and

the hosting SQL Server instance (including the master database). Contained database users can use both Windows and SQL Server authentication.

Implementing the following recommendations and best practices enhances identity and authentication security:

Least-Privilege Role-Based Security:

Utilize least-privilege strategies based on roles to enhance security management.

Place Active Directory (AD) users in AD groups, which should correspond to SQL Server roles. Grant these roles minimal permissions required by the application.

Azure RBAC Controls:

In Azure, adopt least-privilege security using role-based access controls (RBAC).

Active Directory versus SQL Server Authentication:

Prefer Active Directory over SQL Server authentication whenever feasible.

Avoid storing security at the application or database level.

User Lifecycle Management:

Disable accounts promptly when users leave the organization.

Easily remove users from groups during role changes or departures. Group security is a recommended practice.

Multifactor Authentication (MFA):

Implement MFA for accounts with machine-level access, including RDP logins.

MFA mitigates credential theft risks associated with single-factor password-based authentication.

Strong Password Policies

Enforce complex passwords that are not easily guessable.

Regularly update passwords and adhere to Active Directory policies.

Group-Managed Service Accounts (gMSA)

gMSA offer automatic password management, simplified service principal name (SPN) handling, and delegation of management to other administrators.

Key points about gMSA are as follows:

The Windows OS manages passwords for gMSA accounts, eliminating manual administration.

gMSA updates account passwords without service restarts.

It reduces administrative overhead and enhances separation of duties.

Additional security considerations:

Limit rights for DBA AD accounts, restricting access to VMs, OS logins, error logs, and application installations.

Instead of making DBAs sysadmin members, grant CONTROL SERVER permissions. Unlike sysadmin, CONTROL SERVER respects DENY settings.

Data Lineage and Integrity

Maintaining historical records of data changes offers several advantages, including addressing accidental modifications, auditing application changes, and recovering unauthorized data alterations. To achieve this, consider using temporal tables. These tables preserve record versions over time, allowing you to view data as it existed throughout its lifespan and providing a historical perspective for your application's data.

SQL Threats

Common threats to SQL databases encompass a range of security risks that can compromise the confidentiality, integrity, and availability of data. These threats include SQL injection attacks, side channel attacks and other malware threats.

SQL Injection:

SQL injection is a prevalent attack vector where malicious code is injected into input strings that are then passed to an SQL Server for execution. The attacker manipulates input parameters to include additional SQL commands, potentially leading to unauthorized access, data leakage, or even complete control over the database. To mitigate this threat, always use parameterized queries or prepared statements to ensure that user input is properly sanitized and doesn't directly execute SQL commands.

Side-Channel Attacks:

Side-channel attacks exploit information leaked during the execution of a system or application. These attacks don't directly target the software but rather exploit unintended side effects. Examples include timing attacks, where an attacker observes the time taken by a system to respond to different inputs, revealing sensitive information. To defend against side-channel attacks, ensure consistent response times and avoid leaking any information through timing variations.

Malware and Unauthorized Access:

Malware can infiltrate SQL Server environments, compromising data integrity, confidentiality, and availability. Unauthorized access occurs when attackers gain entry to the SQL Server instance or database without proper authentication. Implement robust access controls, regularly update security patches, and use intrusion detection systems to detect and prevent unauthorized access.

Minimizing SQL Injection Risks:

To mitigate the risk of SQL injection, consider the following precautions:

Examine any SQL process that constructs SQL statements to identify potential injection vulnerabilities.

Construct dynamically generated SQL statements using parameterization to prevent injection attacks.

Developers and security administrators should thoroughly review code that involves EXECUTE, EXEC, or sp_executesql.

Avoid using xp_cmdshell due to associated risks. Consider alternatives like SQLCLR.

Disallow certain input characters:

; : Query delimiter

‘ : Character data string delimiter

-- : Single-line comment delimiter

/* … */ : Comment delimiters

Ensure that error outputs do not inadvertently reveal sensitive information to potential attackers.

Minimizing Side Channel Risks:

To mitigate the risk of side-channel attacks, follow these precautions:

Apply the latest patches for both applications and the operating system.

Regularly update firmware for on-premises hardware in hybrid environments.

In Azure, enhance protection for highly sensitive workloads:

Use isolated virtual machines.

Opt for dedicated hosts.

Consider Confidential Compute virtual machines (for example, DC-series or those using 3rd Gen AMD EPYC processors).

Infrastructure Threats

Infrastructure threats in SQL environments encompass risks to the underlying systems and network infrastructure supporting the SQL Server deployments. These threats include vulnerabilities in server hardware, operating systems, network configurations, and cloud infrastructure.

Brute Force Access:

Attackers repeatedly attempt to authenticate using various passwords across different accounts until they find the correct one.

This method relies on sheer persistence and automated tools.

Password Cracking / Password Spray:

In password cracking, attackers use a single, carefully crafted password against all known user accounts.

If the initial attempt fails (password spray), they try different passwords, often waiting between attempts to avoid detection.

Ransomware Attacks:

Ransomware encrypts data and files, rendering them inaccessible.

Attackers demand payment (usually in cryptocurrencies) in exchange for the decryption key.

Common entry points include malicious email attachments or exploit kits targeting software vulnerabilities

Minimizing Ransomware Risks:

Ransomware poses a significant threat to SQL Server environments, targeting databases for encryption or extortion. Attackers exploit vulnerabilities in SQL infrastructure or through phishing attacks to gain access. Once inside, they encrypt database files, demanding payment for decryption. Ransomware can lead to data loss, financial losses, and operational disruptions. Mitigation involves regular backups, strong authentication, patch management, and security awareness training.

To mitigate ransomware risks, consider the following precautions:

Address RDP and SSH Vulnerabilities:

Pay close attention to vulnerabilities in Remote Desktop Protocol (RDP) and Secure Shell (SSH).

Implement security measures to protect against unauthorized access.

Firewalls and Port Lockdown:

Use firewalls to restrict network traffic and lock down unnecessary ports.

Regularly apply security updates to the operating system and applications.

Group Managed Service Accounts

Utilize gMSA for improved security and simplified management.

Access Control for Virtual Machines:

Limit access to virtual machines based on roles and responsibilities.

Implement Just-in-Time (JIT) access and use Azure Bastion for secure remote access.

Surface Area Security:

Avoid installing unnecessary tools (for example, sysinternals, SSMS) on local machines.

Be selective when enabling Windows features, roles, and services.

Regular Backups:

Schedule regular full backups and secure them separately.

Ensure backups are not accessible from common administrator accounts to prevent accidental deletion.

Query Optimization

Query optimization is crucial for improving the performance and efficiency of SQL Server databases. Query optimization directly affects the response time and resource utilization of SQL Server databases. By improving the efficiency of individual queries, the overall performance of the database system is enhanced, leading to multiple benefits like enhanced scalability, improved stability and lower operational costs. Some of the techniques for query optimization on SQL Server 2022 are drafted here:

Use Indexes Identify and create appropriate indexes for columns frequently used in WHERE, and ORDER BY clauses. Avoid over-indexing, as it can lead to increased overhead for data modifications and index maintenance.

Understand Query Execution Use tools like SQL Server Management Studio (SSMS) or SQL Server Extended Events to analyze query execution plans. Understand how SQL Server executes queries and optimize based on the chosen execution plan.

Update Statistics Keep statistics up-to-date to help the query optimizer make informed decisions. Use the UPDATE STATISTICS command or enable the AUTO_UPDATE_STATISTICS option at a database level.

Avoid Table Minimize table scans by ensuring that indexes cover query predicates and join conditions. Use covering indexes to include all columns required for a query to avoid accessing the base table (heap or clustered index).

Parameterize Queries: Try using parameterized queries to promote plan reuse and prevent SQL injection attacks. Parameterized queries help SQL Server generate more efficient execution plans.

Avoid *: Retrieve only the necessary columns in SELECT queries rather than using SELECT Selecting only required columns reduces data transfer overhead and improves query performance.

Optimize Use appropriate join types (for example, INNER JOIN, LEFT based on the relationship between tables. Ensure that join columns are indexed for optimal performance.

Limit the Result Set: Use the TOP clause or ROW_NUMBER() function to limit the number of rows returned by a query. Avoid retrieving unnecessary data to improve query performance.

Avoid Nested Queries and Rewrite nested queries and cursors as set-based operations whenever possible. Set-based operations are generally more efficient than iterative processing.

Consider Table Implement table partitioning for large tables to improve manageability and query performance. Align partitioning with query patterns and index strategies for optimal results.

Use Temp Tables or Table Consider using temporary tables or table variables for complex queries with multiple joins and calculations. Temporary tables can reduce query complexity and improve readability.

Monitor and Analyze Query Use SQL Server Profiler, Extended Events, or Dynamic Management Views (DMVs) to monitor query performance. Identify slow-performing queries and optimize them based on resource usage and execution time.

Avoid Functions Minimize the use of scalar functions in WHERE clauses, as they can prevent index usage. Consider rewriting queries to avoid functions or create computed columns for frequently used calculations.

Use Query Enable Query Store to track query performance over time and identify regressions. Analyze query execution statistics and plan changes to optimize query performance.

Regularly Review and Refactor Queries: Conduct periodic reviews of queries to identify opportunities for optimization. Refactor complex or inefficient queries to simplify logic and improve performance.

Efficient Resource Utilization

Efficient resource utilization in SQL Server has a profound impact on the overall performance, scalability, and stability of the database environment. It also has an impact on cost savings, reliability of the database system and application, better user experience and, facilitates growth and innovation in the organization. The three most important aspects of SQL Server in terms of resource utilization are CPU, memory and IO which have to be maintained well for optimal performance of the environment. Some of the best practices for achieving efficient resource utilization for these three parameters are mentioned as follows:

CPU Utilization

Efficient CPU utilization on SQL Server can be achieved by regularly updating statistics, adding missing indexes, and optimizing high CPU-consuming queries. Monitoring tools like Performance Monitor and SQL Server's DMVs are essential for identifying inefficiencies. Balancing the workload across CPUs and considering hardware upgrades when necessary also contribute to maintaining optimal performance.

Optimize Identify and optimize queries that consume excessive CPU resources through techniques such as index optimization, query rewriting, and reducing unnecessary calculations like Order By clause, and more.

Parallelism Configure maximum degree of parallelism (MAXDOP) appropriately based on server hardware and workload characteristics to prevent excessive parallel query execution and CPU contention.

Resource Governor: Utilize Resource Governor to allocate CPU resources among different workloads or users, ensuring fair distribution and preventing resource contention.

Query Timeout Implement query timeout settings to prevent long-running queries from monopolizing CPU resources and impacting overall system performance.

Memory Utilization

Efficient memory utilization on SQL Server involves setting a maximum server memory limit to prevent excessive consumption, tuning the buffer pool to cache frequently accessed data, and managing the plan cache to optimize query performance. Additionally, enabling the “Lock Pages in Memory” (LPIM) privilege for SQL Server service accounts can prevent memory from being paged out, ensuring consistent performance.

Set Maximum Server Configure maximum server memory to prevent SQL Server from consuming excessive system memory, which could lead to memory pressure and performance degradation. Allocate 80% of the memory to SQL Server thereby leaving 20% for the operating system and other applications.

Buffer Pool Monitor and tune the buffer pool to efficiently cache frequently accessed data in memory, reducing the need for costly disk I/O operations.

Plan Cache Management: Monitor and manage the plan cache to prevent plan cache bloat and optimize plan reuse, improving query performance and reducing memory pressure.

Lock Pages in Memory: Consider enabling “Lock Pages in Memory” (LPIM) privilege for SQL Server service accounts to prevent memory

from being paged out by the operating system, ensuring consistent performance.

I/O Utilization

Efficient I/O utilization on SQL Server involves distributing database files across multiple disks, using SSDs for critical workloads, partitioning tables and indexes, and optimizing TempDB. Regular disk defragmentation, monitoring disk queue length, and efficient backup strategies are also essential.

Storage Optimize storage configuration by distributing database files across multiple disks or storage arrays to balance I/O workload and reduce contention.

Use SSDs for Performance-Critical Workloads: Utilize solid-state drives (SSDs) for storage to improve I/O performance, especially for high-throughput and low-latency workloads.

Partition Tables and Indexes: Implement table and index partitioning to spread I/O workload across multiple files or filegroups, improving manageability and performance for large datasets.

Regularly Defragment Disks: Schedule regular disk defragmentation to optimize disk layout and reduce fragmentation, improving I/O performance and extending disk lifespan.

Monitor Disk Queue Length: Monitor disk queue length to identify I/O bottlenecks and adjust storage configurations or workload distribution accordingly to alleviate contention.

TempDB Optimization: Configure TempDB appropriately by allocating dedicated data files, sizing them correctly, and placing them on fast storage to optimize I/O performance.

Backup and Restore Implement efficient backup and restore strategies to minimize I/O impact during backup and restore operations, ensuring data availability and integrity.

Proactive Performance Monitoring

Proactive performance monitoring is essential for maintaining optimal performance, identifying potential issues before they impact users, and ensuring the smooth operation of SQL Server 2022. Here's a guide to proactive performance monitoring:

Establish Baselines:

Begin by establishing performance baselines for key metrics such as CPU usage, memory usage, disk I/O, and query execution times. Baselines provide a reference point for identifying deviations and trends in performance metrics.

CPU Use Performance Monitor (PerfMon) to track % Processor Time.

Example: Baseline CPU usage might be 30% during peak hours.

Memory Monitor Available MBytes in PerfMon.

Example: Baseline available memory might be 2 GB.

Disk Track Disk Reads/sec and Disk Writes/sec in PerfMon.

Baseline disk read/write might be 1000 reads/sec and 800 writes/sec.

Query Execution Use SQL Server Profiler to capture query durations.

Example: Baseline query execution time might be 200 ms.

Monitor Key Performance Indicators (KPIs):

Continuously monitor critical KPIs such as CPU utilization, memory usage, disk I/O latency, and wait statistics. Utilize built-in performance monitoring tools like SQL Server Management Studio (SSMS), Performance Monitor (PerfMon), or third-party monitoring solutions.

CPU Use SQL Server Management Studio (SSMS) or PerfMon.

Current CPU utilization is 45%.

Memory Monitor using SSMS or PerfMon.

Current memory usage is 75%.

Disk I/O Use PerfMon to track Avg. Disk sec/Read and Avg. Disk sec/Write.

Current disk read latency is 10 ms.

Wait Query DMVs like

Example: High PAGEIOLATCH_SH wait time indicates I/O bottlenecks.

Use Dynamic Management Views (DMVs):

Leverage DMVs to obtain real-time insights into SQL Server performance. Query DMVs to monitor query execution statistics, wait times, and resource utilization.

Query Execution Query sys.dm_exec_query_stats.

Identify queries with high total_worker_time.

Wait Query

Example: High CXPACKET waits indicate parallelism issues.

Resource Query

Example: Check available_physical_memory_kb.

Monitor Query Execution Plans:

Regularly review query execution plans to identify inefficient queries and potential performance bottlenecks. Look for missing or inefficient indexes, excessive table scans, and high CPU or I/O usage in execution plans.

Execution Use SSMS to view execution plans.

Example: Identify missing indexes or table scans.

Inefficient Use sys.dm_exec_query_stats and sys.dm_exec_sql_text.

Example: High total_logical_reads for a query indicates inefficiency.

Set Up Alerts:

Define alert thresholds for critical performance metrics such as CPU usage, memory pressure, and disk latency. Configure alerts to notify administrators when performance metrics exceed predefined thresholds, indicating potential issues.

Alert Use SQL Server Agent to configure alerts.

Example: Alert if CPU usage exceeds 80%.

Configure email or SMS notifications.

Example: Notify DBA if disk latency exceeds 15 ms.

Analyze Historical Data:

Analyze historical performance data to identify long-term trends and patterns. Look for recurring issues, seasonal variations, or performance

degradation over time.

Historical Performance Use SQL Server Data Collector or third-party tools.

Example: Analyze CPU usage trends over the past month.

Recurring Identify patterns in performance data.

Example: Increased load during month-end processing.

Monitor TempDB Usage:

Monitor TempDB usage and contention, as TempDB is a common source of performance bottlenecks. Keep an eye on TempDB space usage, contention on allocation pages, and latch contention.

TempDB Space Query

Monitor

Query sys.dm_exec_requests for latch waits.

Example: High PAGELATCH_UP waits indicate contention.

Track Locking and Blocking:

Monitor locking and blocking activity to identify contention and concurrency issues. Use tools like SQL Server Profiler, Extended Events, or DMVs to capture and analyze locking and blocking events.

Locking and Use SQL Server Profiler or Extended Events.

Example: Capture Lock:Deadlock events.

Analyze Query sys.dm_tran_locks and

Identify blocking sessions.

Review Error Logs and Event Logs:

Regularly review SQL Server error logs and Windows event logs for warnings, errors, and informational messages related to performance issues. Investigate and address issues indicated in the logs to prevent further impact on performance.

Perform Regular Health Checks:

Conduct regular health checks of SQL Server instances to ensure that configurations are optimal and best practices are followed. Review server settings, database configurations, and hardware resources for potential performance bottlenecks.

Health Use SQL Server Best Practices Analyzer (BPA).

Check for configuration issues like max degree of parallelism.

Review Regularly review server and database settings.

Example: Ensure max server memory is appropriately set.

Automate Monitoring Tasks:

Automate performance monitoring tasks using PowerShell scripts, SQL Server Agent jobs, or third-party monitoring tools. Schedule regular checks and reports to ensure consistent monitoring and timely detection of performance issues.

PowerShell Use PowerShell to automate monitoring.

Example: Script to check disk space and send email alerts.

SQL Server Agent Schedule regular monitoring jobs.

Example: Job to capture and log wait statistics daily.

By implementing proactive performance monitoring practices, organizations can identify and address performance issues before they impact users, optimize SQL Server performance, and ensure the efficient operation of their database environment. Regular monitoring, analysis,

and adjustment are essential to maintaining peak performance and reliability.

Conclusion

This chapter explained how to install and configure SQL Server 2022 in different deployment scenarios, such as on-premises, Azure, Linux, or containers, and how to optimize resource allocation, high availability, and security settings. The chapter also covered various strategies for enhancing SQL Server performance, especially in Azure cloud environments, such as dynamic resource scaling, query optimization, indexing, and Azure Monitor integration. It also emphasized on the importance of instance and database level configurations and how a bad configuration impacts the overall stability and performance of SQL Server.

Various security aspects of SQL Server management were discussed, such as authentication modes, encryption, auditing, vulnerability assessment, and threat detection, and provides recommendations for implementing a robust security posture. The chapter illustrates how to use Transparent Data Encryption (TDE) to encrypt data at rest for database files, backup files, and tempdb files, and how to manage encryption keys and certificates. It provides guidance on how to manage identities and authentication for SQL Server, such as using Windows authentication mode, role-based access control, multifactor authentication, group-managed service accounts, and least-privilege principles.

To keep the momentum of this chapter, it is imperative to discuss some of the aspects of performance tuning and optimization and monitoring of

SQL Server in the upcoming chapter. Effective monitoring ensures SQL Server's health, resource utilization, and query performance. Meanwhile, performance tuning involves optimizing queries, indexing, memory management, and I/O subsystems. Regular maintenance tasks contribute to optimal database performance. The next chapter discusses how to manage all these aspects in an efficient and most effective way.

CHAPTER 8

Monitoring, Performance Tuning, and Optimization

Introduction

Monitoring your data estate is crucial for database administrators, developers, or engineers. It allows you to make informed decisions about scaling up, scaling out, or scaling down and keeps you aware of any issues that may arise in your data layer. Additionally, tuning your data layer for optimal performance is essential for ensuring that your applications can effectively access and work with data. This chapter will explore important tips and tricks for achieving these goals across the SQL realm.

Structure

The topics that we cover in this chapter are:

Monitoring SQL Server on Windows, Linux, Containers and Across Azure SQL

Learning the Subtle Art of Troubleshooting SQL Server

Enable the SQL Server Self-Healing Power

Optimize SQL Server on Azure

Monitoring

A well-designed and carefully implemented monitoring solution is essential for maintaining a healthy and functional database infrastructure. It keeps system and database owners informed about the status of the infrastructure and provides accurate information that enables effective decision-making. About 60% of monitoring requirements are standard across all customers, while the remaining 40% require customization to meet specific needs. These needs depend on the business vertical and application requirements.

For instance, some customers prioritize monitoring specific reports and functions that must run within a certain time frame. For others, ensuring high availability of the database across geo-regions is more important than performance. And for some, both high availability across geo-regions and performance are essential for their application's operation.

When selecting a suitable monitoring tool for your environment, you have several options to consider: commercial products, custom-built solutions, or open-source solutions. Regardless of the path you choose, here are some crucial factors to weigh, ensuring that your decision is well-informed.

Understand your data When creating a strategy for establishing or enhancing an existing monitoring infrastructure, it becomes crucial to realize both your current data infrastructure and anticipate future requirements. For instance, consider the following:

Currently, your SQL Server instances may be distributed across on-premises data centers in various regions. However, as part of your data infrastructure modernization, you might contemplate expanding these SQL Server instances to cloud infrastructure. In this context, you could opt for deployment options such as Azure SQL Database, Azure SQL Managed Instance, SQL Server on Azure VMs, or SQL Server instances in non-Azure cloud environments.

Additionally, you might be considering deploying SQL Server on various platforms, including SQL on Linux, Windows, containers, or Kubernetes-based environments.

In your environment, you might encounter a mix of database platforms such as Microsoft SQL Server, MySQL, PGSQL, or Oracle. Therefore, when selecting a monitoring solution, ensure that it is capable of monitoring most of these database platforms.

Therefore, it is essential to select a solution that works effectively across most of these scenarios. This ensures that you avoid the complexity of collecting monitoring data from disparate sources.

A flexible and scalable monitoring solution has the potential to generate substantial data volumes that require careful management and analysis. Consequently, performance optimization becomes a critical aspect of the design. Ensure that you allocate sufficient resources—such as CPU, memory, and I/O—to the monitoring platform and prioritize scalability.

Redundancy: To achieve a successful monitoring platform, it is crucial to avoid single points of failure. Therefore, prioritize high availability and disaster recovery for the overall platform. This ensures that if the solution

becomes unavailable in one region, there is an alternative region ready to provide monitoring information within an acceptable timeframe.

Easy onboarding of new Your data infrastructure is dynamic, and over time, you will introduce new SQL Server instances while retiring old servers. Therefore, your monitoring solution should be adaptable for easy onboarding and offboarding of new targets, allowing seamless data collection adjustments.

Data Collection, Retention, and Access Keep in mind that the monitoring infrastructure accumulates substantial data. Therefore, it is crucial to establish processes and utilize technology to manage and control the data collection from each server. Determine the duration for which data should be retained within the system. Additionally, implement controls to specify which individuals within your organization can access the data, dashboards, and reports.

Alerting Options: A robust monitoring infrastructure should also possess the capability to send alerts when specific conditions are triggered. To achieve this, you need well-defined processes and systems in place to identify the individuals who should receive these alerts. Notifications can be delivered via email, phone calls, or messages to mobile devices.

While essential considerations for selecting a monitoring solution have been covered, here is a monitoring solution for SQL Server that aligns with many of the discussed points. This is a sample example of configuring your own custom monitoring solution. The critical components used in this solution are Telegraf, InfluxDB, and Grafana. The entire solution is container-based, hence providing the agility to deploy and scale based on the requirements.

Telegraf, an open-source agent, is utilized for establishing connections with SQL Server instances to gather data. It is commonly used for the collection and reporting of metrics, capable of monitoring a variety of sources, including system, application, and custom metrics. With support for over three hundred plugins, Telegraf can collect from diverse sources and write to different destinations. For detailed information, refer to the Telegraf repository on GitHub. In the given scenario, Telegraf's input plugin is tasked with collecting data from SQL Server, while the output plugin is responsible for writing this data to InfluxDB.

InfluxDB is a specialized open-source database tailored for the efficient handling and retrieval of time series data, ensuring high availability and speed. It is particularly suited for recording data at regular intervals over time, making it ideal for monitoring changes, trends, and pattern analysis. For further details, you can check the InfluxData repository on GitHub.

Having gathered data via Telegraf and stored it in InfluxDB, a time series database, the next step is to employ a platform for dashboard creation and data visualization. Grafana serves this purpose in our example monitoring solution. It enables querying, visualizing, alerting, and exploring data from various sources, including InfluxDB. In this instance, users will utilize Grafana to view dashboards and visualize the data collected. Grafana is also an open-source platform.

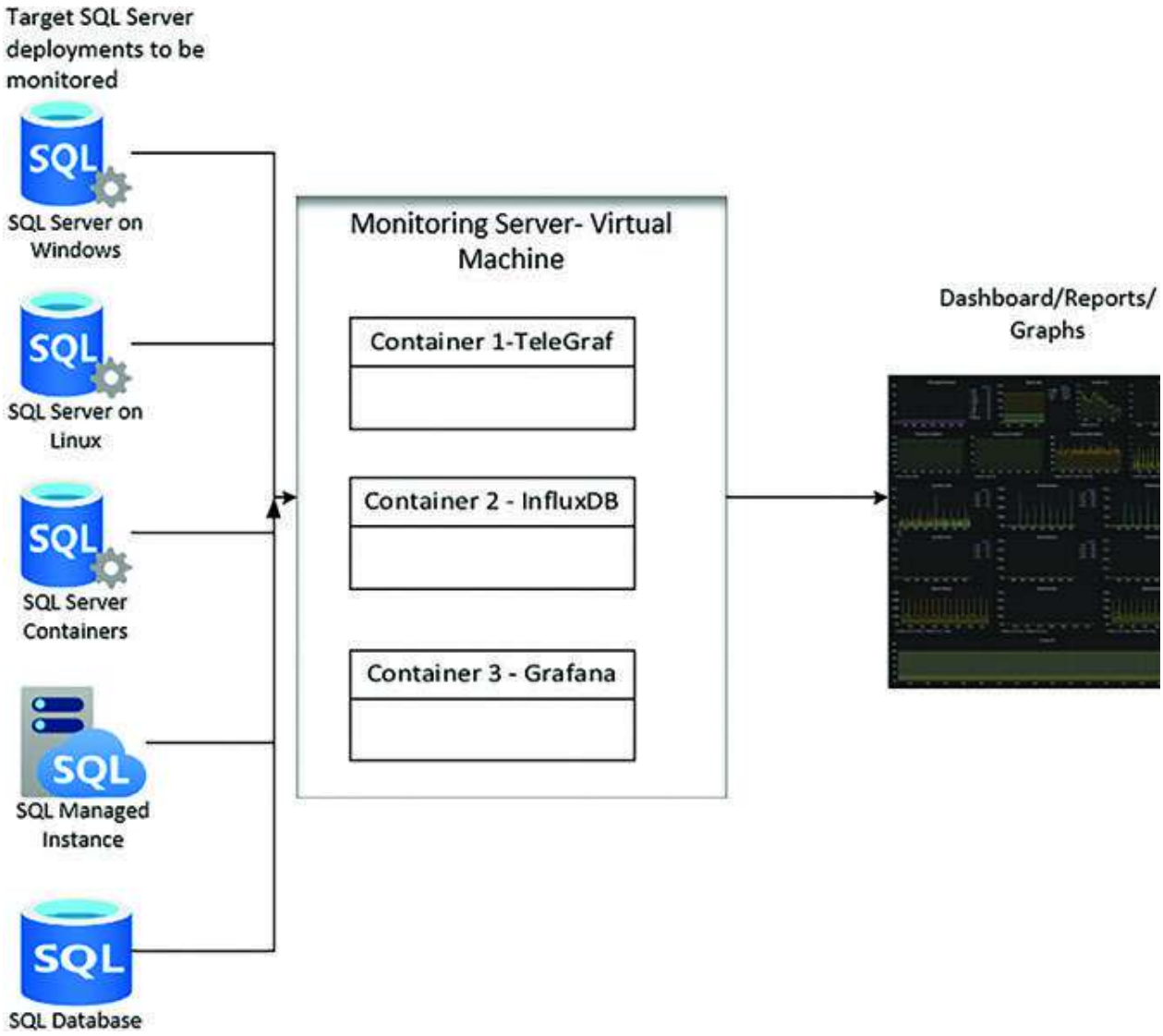


Figure 8.1: A sample monitoring tool for SQL Server deployments across on-premises and Azure SQL

In Figure we observe that this solution facilitates data collection from diverse SQL Server targets. These targets span SQL Servers deployed either on-premises or within the Azure SQL family. Furthermore, the tool extends its monitoring capabilities to SQL Server instances running on Windows, Linux, or containers—regardless of whether they are deployed on-premises, in Kubernetes, on Virtual Machines, or in the Cloud.

For this demonstration, we will be utilizing a single virtual machine that hosts all three containers. However, depending on the number of instances you monitor and the data collected, you may choose to distribute these containers across multiple virtual machines.

As you may have already followed, we are utilizing containers and virtual machines because they facilitate agile resource scaling—specifically, for CPU and memory. This flexibility allows us to easily adjust our scaling based on the specific needs. Also, the monitoring server that is used for the demonstration is an Ubuntu-based VM. You can run this on your choice of distribution.

Before we start the deployment of this sample monitoring solution, let us setup the pre-requisites needed for this solution.

For this demonstration, an Azure Ubuntu-based Virtual Machine will be used to establish an Ubuntu-based server. The containers can be run on any desired distribution.

Then we install docker and create a common Docker network for all containers that we intend to deploy.

InfluxDB, Grafana, and Telegraf containers are all being deployed on a single node for this presentation.

Once you have the VM deployed and ready, login to the VM and run the following command to create the common Docker network:

#You can change the name of the network from “influxdb-telegraf-net” to whatever you want.

docker network create --driver bridge influxdb-telegraf-net

You can list the network using the following command, and you should see the new network that you created in the step before.
docker network ls

Now, on the target SQL Servers, the SQL Server instances that you intend to monitor, you need to prepare the instance to be monitored by creating a login and giving the login enough permissions to query the DMVs and other system functions to help collect the monitoring data. For this demo, the login name that we will create on the target SQL Server instances is called telegraf.

#You need to run these commands on all the SQL Server instances that you intend to monitor
USE master;
CREATE LOGIN telegraf WITH PASSWORD = N'StrongPassword1!',
CHECK_POLICY = ON;
GO
GRANT VIEW SERVER STATE TO telegraf;
GO
GRANT VIEW ANY DEFINITION TO telegraf;
GO

Let us deploy the containers using the following commands:

to deploy the telegraf container, run the following command:
where:/home/amvin/monitor/sqltelegraf/telegraf.conf is a telegraf configuration file placed on # my host machine, please update the path as per your environment.

please ensure that you change the IP addresses and port numbers to your target SQL Server #instances in the telegraf.conf file that you create in your environment.

docker run -d --name=telegraf -v /home/amvin/monitor/sqltelegraf/telegraf.conf:/etc/telegraf/telegraf.conf --net=influxdb-telegraf-net telegraf

#deploy the Influx DB containers.
where: /home/amvin/monitor/data/influx is a folder on the host that we are mounting inside the #container, you can create this folder in any location.
please ensure you set the right permissions so files can be written inside this folder by the #container.
docker run --detach --net=influxdb-telegraf-net -v /home/amvin/monitor/data/influx:/var/lib/influxdb:rw --hostname influxdb --restart=always -p 8086:8086 --name influxdb influxdb:1.8
#Finally deploy the Grafana-based container:
docker run --detach -p 3001:3000 --net=influxdb-telegraf-net --restart=always -v /home/amvin/monitor/data/grafana:/var/lib/grafana -e "GF_INSTALL_PLUGINS=grafana-piechart-panel,savantly-heatmap-panel" --name grafana grafana/grafana:8.1.1
where: home/amvin/monitor/data/grafana is a folder on the host that we are mounting inside the container, you can create this folder in any location.
please ensure you set the right permissions so files can be written inside this folder.
grafana-azure-monitor-datasource is already included with grafana, so removing it from the list of plugins to install.

Like we discussed earlier, every monitoring solution should have a data retention policy, and our solution also provides this capability. Let's now

setup the retention policy on InfluxDB. In this demo, we are setting this to 30 days; you can change this based on your organizational requirement.

#We are going to login into the InfluxDB container to create a database telegraf, which will be used to store the data, and then we are also going to set the retention policy as follows:
sudo docker exec -it influxdb bash
# then run the following commands inside the container
influx
create database telegraf;
use telegraf;
show retention policies;
create retention policy retain30days on telegraf duration 30d replication 1 default;
quit

Time to now setup the Grafana and create the dashboard that displays the data collected by telegraf, stored by InfluxDB.

Browse to your Grafana instance:
http://[GRAFANA_IP_ADDRESS_OR_SERVERNAME]:3000

First time you login to Grafana, the login and password are set to: admin.
Also, take a look at the Getting Started Grafana documentation.

Add a data source for InfluxDB. The detailed instructions are in the grafana data source docs

InfluxDB

InfluxDB (this is also the default)

http://[INFLUXDB_HOSTNAME_OR_IP_ADDRESS]:8086. (The default of http://localhost:8086 works if Grafana and InfluxDB are on the same machine; make sure to explicitly enter this URL in the field.)

telegraf

Click “Save & Test.” You should see the message “Data source is working.”

Download Grafana dashboard JSON definitions from the repo here at: https://github.com/microsoft/mssql-docker/blob/master/linux/monitor/dashboard/dashboard.json Then, import them into Grafana. You can refer to the link to perform the import: https://grafana.com/docs/grafana/latest/dashboards/build-dashboards/import-dashboards/#importing-a-dashboard

You are ready, and Figure 8.2 shows how the dashboard should look. Feel free to modify the graphs as per your requirement. That’s it, hope this helps. The entire code is available on the book’s GitHub repo for quick reference: SQL-Server-Unveiled---Path-to-Data-Excellence/Chapter 8 at main · ava-orange-education/SQL-Server-Unveiled---Path-to-Data-Excellence (github.com)

Figure 8.2: Grafana dashboard showing monitoring data from the SQL Server instances

Monitoring for Azure SQL Database

Monitoring the performance of Azure SQL Database is crucial to ensure optimal performance, identify bottlenecks, and troubleshoot issues. Azure provides various tools and features to help you monitor the performance of your SQL databases. Here is a detailed explanation of Azure SQL monitoring for performance:

Azure Portal

Metrics:

Azure SQL Database provides various metrics that you can monitor through the Azure Portal. These metrics include CPU usage, storage metrics, Database Transaction Unit (DTU) consumption, and more. These metrics give you insights into the performance, resource consumption, and health of your database. Here is a detailed explanation of some key metrics in the Azure Portal for Azure SQL Database, along with examples:

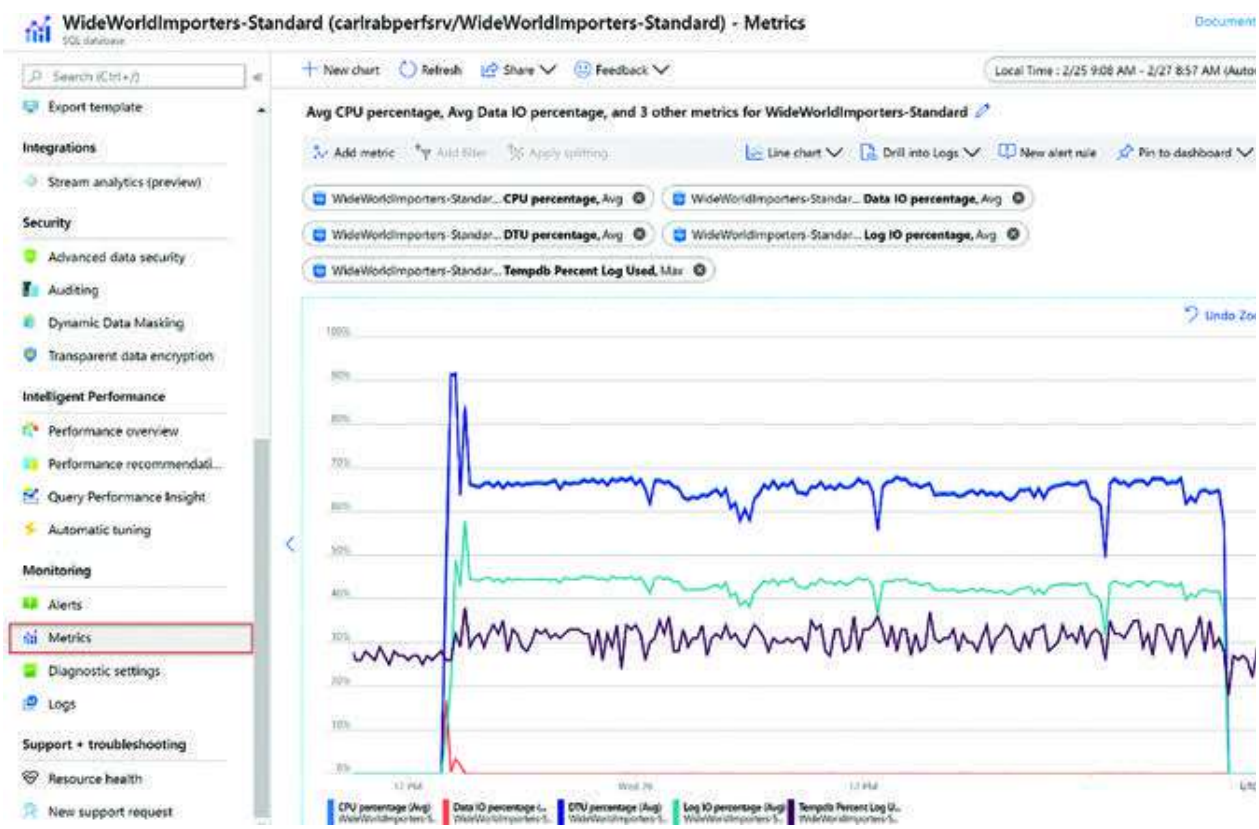


Figure 8.3: Metrics View under Azure Monitoring

Source: https://learn.microsoft.com/en-us/azure/azure-sql/database/monitor-tune-overview?view=azuresql

Database Transaction Unit (DTU) Metrics:

Indicates the percentage of the provisioned Database Transaction Units (DTUs) being consumed by the database. It helps you understand the overall workload on your database.

If you have provisioned 100 DTUs and the consumption percentage is consistently high (example, 80%), it indicates that your database is utilizing most of the allocated resources.

CPU Metrics:

Represents the percentage of CPU utilization by the SQL Database. High CPU usage might indicate resource contention or inefficient queries.

If the CPU percentage is consistently above 90%, you may need to optimize queries, add indexes, or consider scaling resources.

Storage Metrics:

Displays the amount of storage used by the database. Monitoring storage is crucial for capacity planning and avoiding storage-related issues.

If your database is approaching its storage limit, you might need to clean up unnecessary data, archive old records, or consider increasing storage capacity.

Memory Metrics:

Indicates the percentage of memory utilized by the SQL Database. Monitoring memory usage is essential for optimal query performance.

If the memory percentage is consistently high, it may impact query performance. Review and optimize queries, and consider scaling resources if necessary.

Throughput Metrics:

Represents the percentage of log write throughput. It helps in monitoring the efficiency of write operations to the transaction log.

A sudden increase in log write percentage might indicate high write activity. Investigate and optimize write-intensive queries or transactions.

Query Store Metrics:

Indicates the percentage of Query Store storage used. Query Store captures query performance data and execution plans.

If the Query Store storage is consistently high, it may impact performance. Consider purging old data or adjusting the Query Store retention period.

Resource Utilization Metrics:

Displays the percentage of resources used by the database. This metric gives an overall view of resource consumption.

If the resource utilization percentage is high, it suggests that your database is approaching its resource limits. Consider scaling resources to meet demand.

Deadlocks Metrics:

deadlocks: Indicates the number of deadlocks encountered by the database. Monitoring deadlocks helps identify and resolve concurrency issues.

An increase in the number of deadlocks may require reviewing and optimizing transaction isolation levels or redesigning queries.

Wait Statistics Metrics:

Provides insights into the types of resource waits experienced by the database. Identifying and addressing wait types can improve overall performance.

If you observe high wait times related to storage, it may indicate I/O bottlenecks. Optimize queries or consider storage performance improvements.

Auto-Tuning Metrics:

Indicates the number of successful automatic tuning actions performed by the database engine, such as creating or dropping indexes.

If you see a high number of successful actions, it indicates that automatic tuning is actively optimizing your database for better performance.

Azure SQL Database Advisor Metrics:

Shows the number of recommendations provided by Azure SQL Database Advisor. Recommendations cover performance, security, and best practices.

Regularly review the number and types of recommendations to implement best practices and optimize your database's performance.

These are just a few examples of the many metrics available in the Azure Portal for Azure SQL Database. Regularly monitoring these metrics and responding to anomalies or performance bottlenecks is essential for maintaining a healthy and high-performing database environment. Adjusting resource configurations, optimizing queries, and following best practices based on the insights gained from these metrics contribute to the overall success of your SQL Database deployment.

Azure Monitor

You can use Azure Monitor to create custom dashboards and set up alerts based on specific performance metrics. This allows you to proactively address potential issues. Azure Monitor is a comprehensive monitoring solution that allows you to collect, analyze, and act on telemetry data from your Azure resources. It includes features such as Metrics, Logs, Application Insights, and Alerts. Here is a detailed explanation of using Azure Monitor for Azure SQL Database:

Create Custom Dashboards:

Azure Monitor enables you to create custom dashboards to visualize and monitor specific metrics for your Azure SQL Database. These dashboards can be tailored to display the most relevant information for your monitoring needs.

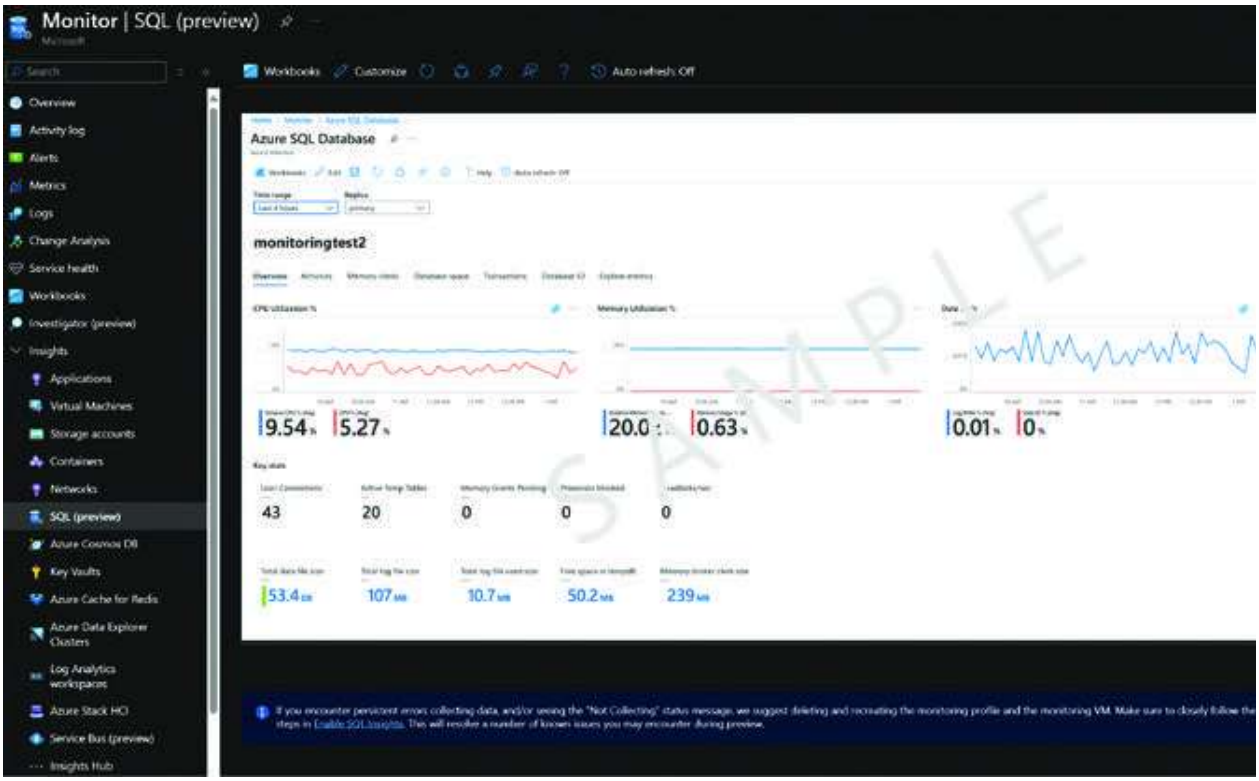


Figure 8.4: Metrics View under Azure Monitoring

Example: Creating a Custom Dashboard for Azure SQL Database Metrics:

Navigate to the Azure Portal.

In the left sidebar, click “Monitor.”

Under “Monitoring solutions,” select “Metrics.”

Choose your Azure SQL Database as the resource.

Select specific metrics such as CPU percentage, DTU consumption, and storage metrics.

Click “Add to dashboard” to create a new dashboard or add to an existing one.

Set Up Alerts:

Azure Monitor allows you to configure alerts based on specific conditions, thresholds, or anomalies in your Azure SQL Database metrics. This helps you proactively address potential issues by receiving notifications when predefined conditions are met.

Example: Setting Up an Alert for High CPU Usage:

In the Azure Portal, go to “Monitor” and select your Azure SQL Database.

Under “Settings,” choose “Alerts.”

Click “New alert rule.”

Configure the alert rule by selecting the target resource, condition (e.g., CPU percentage > 90%), and threshold.

Specify the action group to define who receives the alert notifications.

Save the alert rule.

View Metrics and Logs:

Azure Monitor provides a unified view of metrics and logs for your Azure SQL Database. You can explore historical data, analyze trends, and troubleshoot issues using these metrics and logs.

Example: Analyzing Query Performance Using Logs:

Navigate to the Azure Portal and select your Azure SQL Database.

Under “Settings,” choose “Logs” to access Azure Monitor Logs.

Run queries to analyze logs related to query performance, deadlocks, or other diagnostic information.

Use Kusto Query Language (KQL) to filter and aggregate data for deeper analysis.

Application Insights Integration:

Azure Monitor integrates with Application Insights, providing end-to-end monitoring for your applications and databases. This integration allows you to correlate performance data from your application and database for a holistic view.

Example: Correlating Application and Database Performance:

Enable Application Insights for your application.

In the Azure Portal, navigate to “Application Insights” and select your application.

Explore the “Performance” tab to view both application and database performance metrics in one place.

Identify correlations between application events and database queries.

Azure Advisor Integration:

Azure Monitor is integrated with Azure Advisor, which provides intelligent recommendations for improving the performance, security, and reliability of your resources.

Example: Implementing Azure Advisor Recommendations:

In the Azure Portal, go to “Advisor.”

Review recommendations related to your Azure SQL Database.

Implement suggested actions, such as optimizing queries, adjusting resource configurations, or applying security best practices.

In summary, Azure Monitor empowers you to tailor your monitoring strategy, visualize relevant metrics, set up proactive alerts, and integrate insights from various sources. Utilizing custom dashboards, alerting mechanisms, and integrations with other Azure services, you can effectively monitor and optimize the performance of your Azure SQL Database. Regularly reviewing and adjusting these configurations based on evolving requirements ensures a proactive approach to managing your database resources.

Query Performance Insights

Query Performance Insight provides intelligent query analysis for single and pooled databases. It helps identify the top resource-consuming and long-running queries in your workload, allowing you to optimize performance and efficiently utilize the resources you are paying for.

Query Performance Insight (QPI)

Query Performance Insight (QPI) is a feature within Azure SQL Database designed to help users identify and optimize poorly performing queries. It provides detailed insights into the resource consumption of individual queries, allowing for targeted optimization efforts.

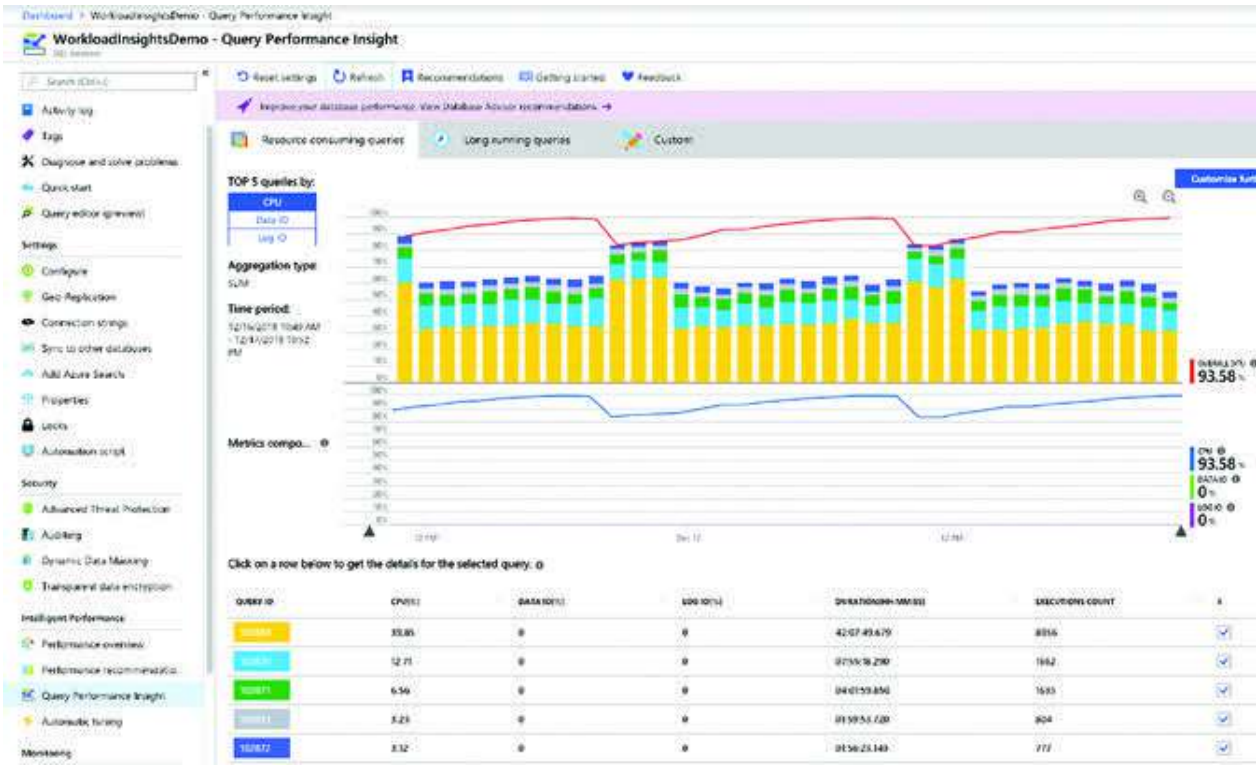


Figure 8.5: Query Performance Insights

Source: https://learn.microsoft.com/en-us/azure/azure-sql/database/query-performance-insight-use?view=azuresql

QPI is accessible through the Azure Portal for Azure SQL Database. Navigate to the Azure SQL Database instance in the Azure Portal. Under the "Intelligent Performance" section, go to "Query Performance Insight." QPI offers a graphical representation of the top resource-consuming queries over a specified time period. It provides information on CPU usage, duration, and execution count for each query. Queries that consume a significant amount of resources (CPU or duration) are highlighted. Queries can be sorted based on different metrics to identify the most resource-intensive ones. Some of the key metrics defined in QPI are:

CPU Usage: Indicates the amount of CPU resources consumed by a specific query.

Duration: Represents the total time taken by a query to execute.

Execution Count: Displays how many times a particular query has been executed.

QPI allows users to select a specific time range for analysis, helping users to focus on recent or historical query performance. The time range can be adjusted to identify trends or spot specific incidents of poor query performance. Clicking on a specific query in the QPI interface will show detailed page for that query. The detailed page provides a breakdown of resource consumption over time, execution plans, and additional statistics.

QPI not only highlights poorly performing queries but also offers optimization recommendations. Recommendations may include creating or

modifying indexes, rewriting queries, or other performance-enhancing actions.

QPI maintains a history of query performance, which allows tracking changes over time. Historical data helps in understanding the impact of changes to the database schema, indexes, or query patterns. QPI works seamlessly with Query Store, another feature in Azure SQL Database. Query Store captures query plans and performance metrics, providing additional context for query analysis. QPI facilitates a proactive approach to query optimization by allowing users to address performance issues before they impact the overall database performance.

Example Scenario:

Suppose you observe a spike in CPU usage during a specific time range. By accessing QPI and analyzing the top resource-consuming queries during that period, you identify a poorly optimized query responsible for the increased CPU utilization. QPI not only highlights the query but also provides recommendations for optimization, such as creating an index. Implementing the suggested optimizations results in a significant reduction in CPU usage and an overall improvement in database performance.

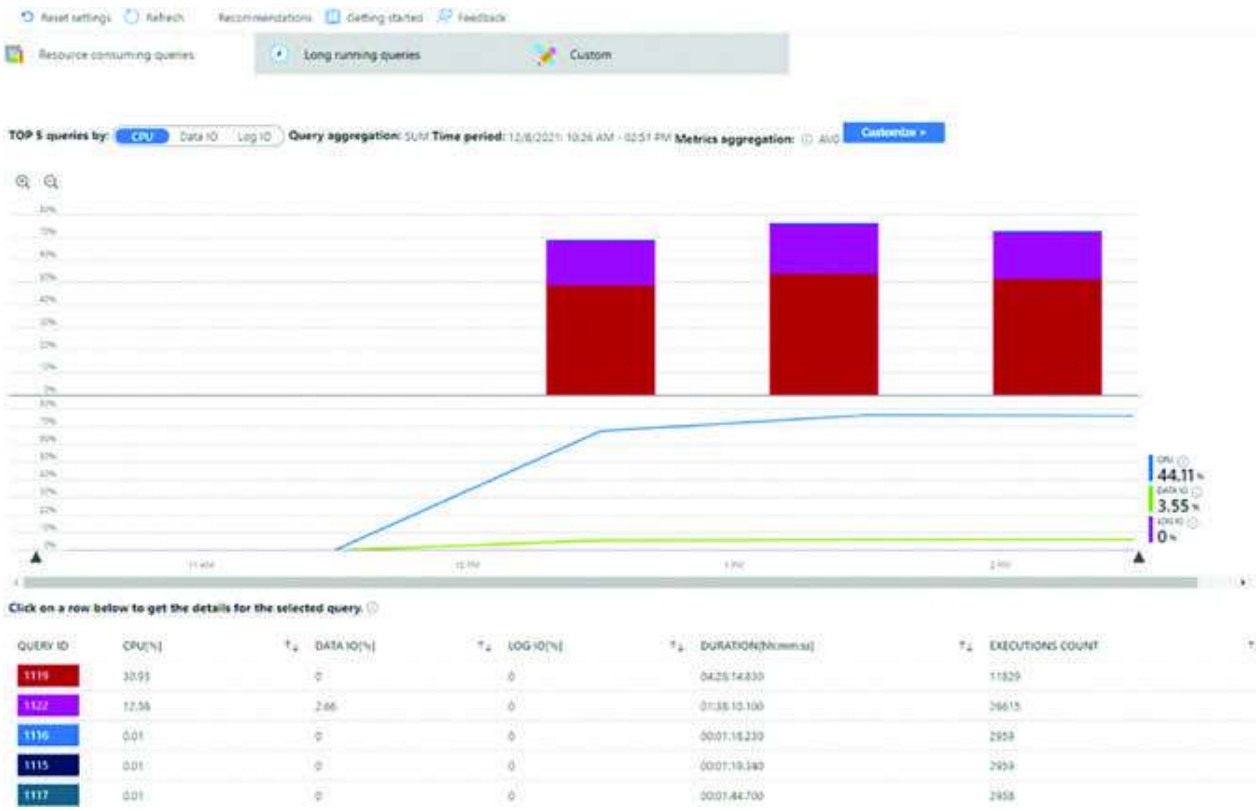


Figure 8.6: Query Performance Insights - CPU Usage Metrics

Source: https://learn.microsoft.com/en-us/azure/azure-sql/database/high-cpu-diagnose-troubleshoot?view=azuresql

In summary, Query Performance Insight is a valuable tool in the Azure SQL Database ecosystem, empowering users to identify, analyze, and optimize poorly performing queries for better overall database performance. Regularly utilizing QPI as part of your performance monitoring strategy ensures a proactive approach to query optimization and resource management.

Query Store: Query Store allows you to capture query execution statistics and execution plans over time, facilitating performance analysis and troubleshooting. Query Performance Insight relies on data captured by the Query Store. If the database has no activity or if the Query Store was inactive during a specific period, the charts in Query Performance Insight will appear empty for that time range.

Query Store captures information such as executed queries, execution plans, runtime statistics, and wait statistics. Historical data is stored in the database, allowing for trend analysis and performance troubleshooting. It provides metrics such as execution count, CPU time, logical and physical reads, and duration for each query. These metrics help in identifying resource-intensive queries and understanding their impact on the database. It becomes quite easy to identify performance regressions by comparing query performance over different time periods. The regressed queries could be quickly identified in terms of resource consumption or execution time. Query Store also allows forcing a specific execution plan for a query. This is beneficial when an execution plan is identified that performs well, and DBA wants to ensure it is consistently used.

In Azure SQL Database, Query Store is integrated with Automatic Tuning. This feature can automatically apply performance-improving actions, such as creating or dropping indexes, based on the data collected by Query Store. The historical data stored in Query Store is invaluable for analyzing the performance of queries over time. DBAs can easily identify trends, understand changes in workload, and correlate performance issues with specific queries. In SSMS, Query Store provides reports and insights into the top resource-consuming queries, making it easy to prioritize optimization efforts. Users can focus on queries that have the most significant impact on database performance and even compare execution plans for a specific query across different time intervals. This is useful for understanding changes in query plans and their impact on performance.

The following figure depicts the performance statistics such as and Logical Reads for a database over the last month with the help of Query Store dashboard in SSMS.

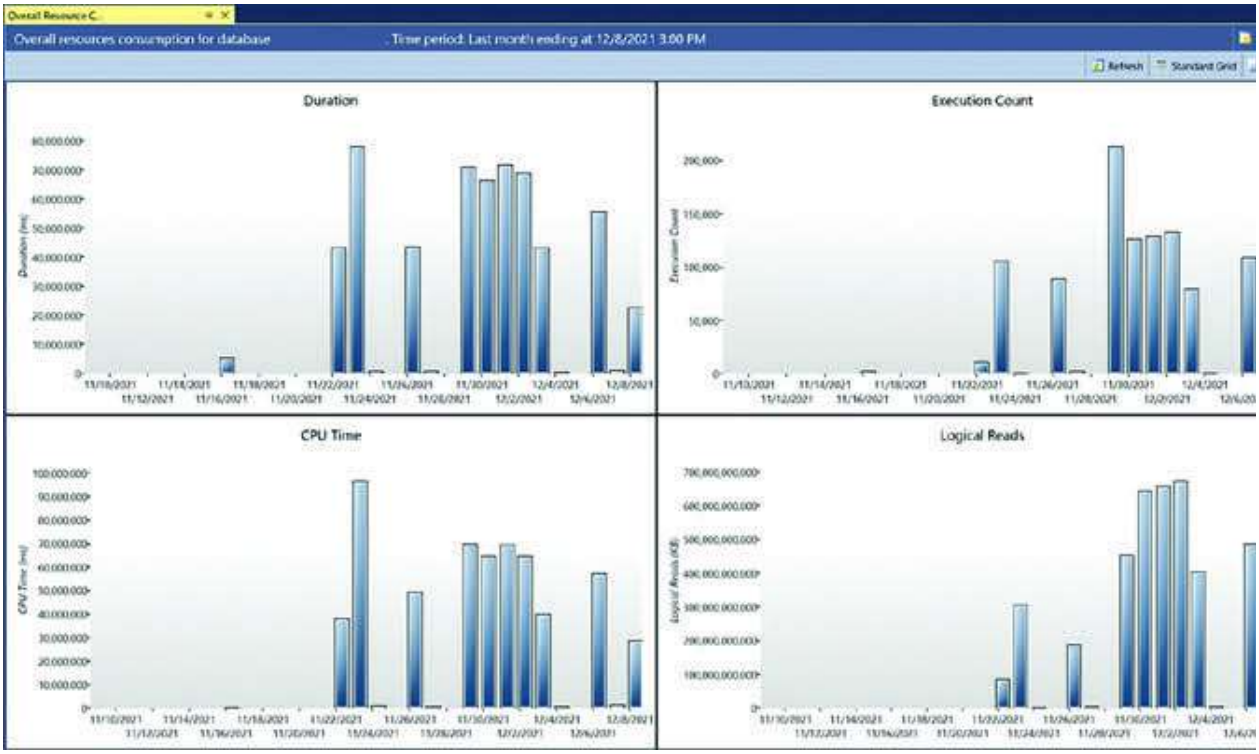


Figure 8.7: Query Store - Historic CPU Utilization
Source: https://learn.microsoft.com/en-us/azure/azure-sql/database/high-cpu-diagnose-troubleshoot?view=azuresql

Automatic Tuning

Automatic Tuning is a feature in Azure SQL Database that leverages artificial intelligence and machine learning to optimize the performance of your database. It automates the implementation of performance-enhancing actions, such as creating or dropping indexes, based on the historical performance data and query workload.

Automatic Tuning is enabled by default in Azure SQL Database, and it can be configured for individual databases. It can be enabled or disabled Automatic

Tuning through the Azure Portal or using Transact-SQL commands. Automatic Tuning continuously monitors the performance of the database and generates performance recommendations. These recommendations are based on the historical execution plans, index usage, and other relevant metrics. Automatic Tuning currently supports two types of actions: creating and dropping indexes. When enabled, it can automatically create missing indexes to improve query performance or drop redundant indexes that are not being used.

Automatic Tuning recommendations are integrated with the Azure SQL Database Performance Dashboard. The dashboard provides an overview of the tuning actions taken and their impact on database performance.

When Automatic Tuning applies an action, it generates alerts to notify administrators about the changes made to the database. Alerts include information about the action taken, the affected index, and the performance improvement achieved. While Automatic Tuning is designed to work autonomously, database administrators can manually override its decisions. If needed, administrators can review recommendations, choose not to apply them, or implement alternative solutions.

Automatic Tuning actions are logged and can be reviewed in the Azure Portal or through Transact-SQL. Database administrators can analyze the impact of tuning actions on query performance and adjust their strategies accordingly. Automatic Tuning includes a rollback mechanism in case a tuning action negatively impacts performance. If a performance regression is detected, Automatic Tuning can automatically revert the changes to the previous state.

Automatic Tuning complements traditional database optimization practices but does not replace the need for careful database design and query optimization. It is recommended to regularly review and understand the tuning actions taken by Automatic Tuning to ensure they align with your

performance goals. Let us now look at some of the use cases that would leverage the Automatic Tuning feature.

Use Case 1: Index Creation for Improved Query Performance

The database workload changes over time, and a frequently executed query lacks a necessary index, resulting in suboptimal performance.

Automatic Tuning Action: Automatic Tuning identifies the missing index and automatically creates it to improve query performance.

Result: The query now benefits from the newly created index, leading to reduced query execution time and overall improved database performance.

Use Case 2: Index Drop for Resource Optimization

Scenario: An index that was once useful is no longer utilized by queries, contributing to unnecessary storage consumption and maintenance overhead.

Automatic Tuning Action: Automatic Tuning detects the unused index and automatically drops it to optimize resource utilization.

Result: Storage space is reclaimed, and database maintenance operations become more efficient without the overhead of unnecessary indexes.

The following figure depicts automatic tuning settings where Force Plan option is ON, Create Index option is OFF, and Drop Index option is set to ON.

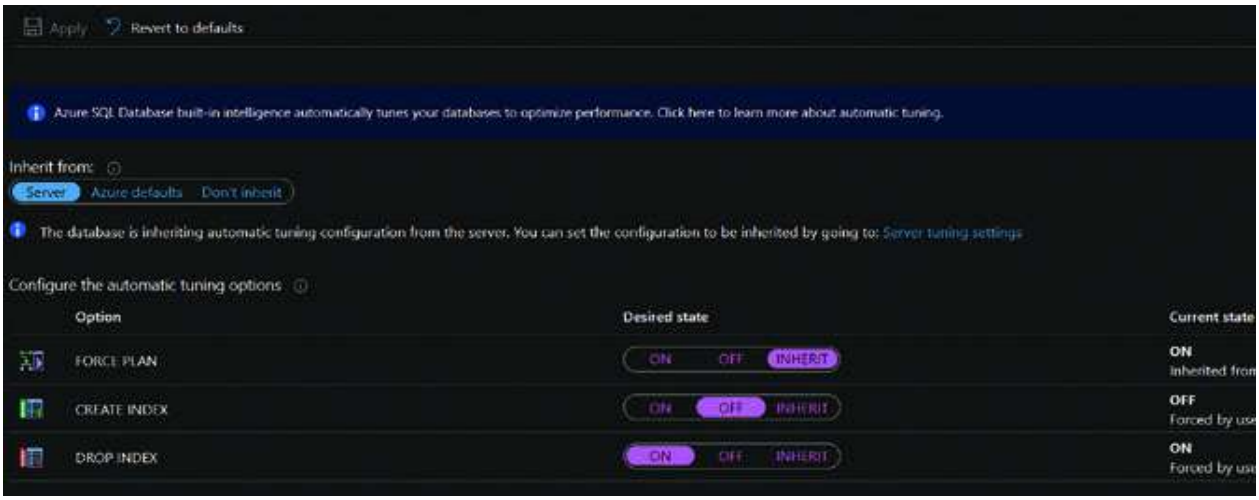


Figure 8.8: Automatic Tuning Blade

In summary, Automatic Tuning in Azure SQL Database automates performance optimization actions based on historical data and workload patterns. It enhances the adaptability and efficiency of database performance management, ensuring that your database continually evolves to meet the demands of changing workloads. Regular monitoring and analysis of Automatic Tuning actions contribute to a proactive approach to performance tuning and optimization.

Performance Recommendations

Azure SQL Database Performance Recommendations is a feature that provides intelligent insights and suggestions to improve the performance, security, and reliability of your Azure SQL Database. These recommendations are based on the analysis of historical performance data, usage patterns, and best practices. Here is a detailed explanation of Azure SQL Database Performance Recommendations:

Azure SQL Database Performance Recommendations cover various areas, including performance, security, and best practices. Examples of

recommendation categories include indexing, query optimization, security configurations, and database design. Each recommendation provides detailed information about the suggested action, the potential impact, and steps to implement the recommendation. Users can review the information to make informed decisions about applying the recommendations. Performance Recommendations include an analysis of the potential impact of implementing the suggested actions. This helps users assess the benefits and trade-offs associated with each recommendation.

Many recommendations align with best practices for Azure SQL Database. Implementing these recommendations ensures that your database follows industry standards for performance and security. Performance Recommendations provide proactive guidance for optimizing database performance and addressing potential issues before they impact the application. Performance Recommendations are dynamic and can change based on workload patterns and changes in the database environment. Regularly monitoring and reviewing recommendations ensures that your database continually benefits from optimization suggestions.

Here are some scenarios which would utilize Performance Recommendations feature:

Scenario 1: Indexing Recommendation

Scenario: A frequently executed query lacks a necessary index, leading to suboptimal performance.

Recommendation: Performance Recommendations suggest creating a specific index to enhance the query performance.

Result: Implementing the recommended index improves the query's execution time, leading to better overall database performance.

Scenario 2: Query Optimization Recommendation

Scenario: A complex query experiences high CPU usage and slow execution.

Recommendation: Performance Recommendations suggest rewriting the query or adding missing joins for optimization.

Result: Implementing the query optimization recommendations reduces resource consumption and improves query performance.

The following figure depicts Performance Recommendation for a sample database:

Figure 8.9: Performance Recommendation Blade

Source: https://learn.microsoft.com/en-us/azure/azure-sql/database/database-advisor-find-recommendations-portal?view=azuresql

In summary, Azure SQL Database Performance Recommendations offer valuable insights and guidance for optimizing database performance, security, and adherence to best practices. Regularly reviewing and implementing these recommendations contributes to a proactive approach to managing and improving the health of your Azure SQL Database.

Alerting and Notifications

Azure SQL Database provides robust alerting and notification capabilities to help you proactively monitor and respond to critical events and conditions. Alerts can be configured based on specific performance metrics, resource usage, and other criteria.

Alert rules allow you to receive notifications when specific metrics fall outside expected ranges. You can customize the scope of an alert rule to meet your requirements. For instance, the scope can be set to monitor a single metric or multiple metrics:

A single database

An elastic pool

All databases or elastic pools in a resource group

All databases or elastic pools in a subscription within an Azure region

All databases or elastic pools in a subscription within all regions

Alert rules periodically assess aggregated metric values over a specified lookback period, comparing them against a predefined threshold. You have the flexibility to configure the threshold value, evaluation frequency, and the duration of the lookback period.

When an alert rule is triggered, you receive notifications based on your specified preferences within the associated action group. These preferences can include email notifications, SMS messages, voice alerts, or other custom actions like webhooks, automation runbooks, and logic apps. Additionally, you can seamlessly integrate alerts with supported IT service management tools.

The choice of metrics and the most effective thresholds for alert rules can differ significantly based on the diverse range of customer workloads in Azure SQL Database. The suggested alerts in the table serve as a foundation to assist you in shaping the ideal alerting setup for your Azure SQL Database resources. However, your specific configuration may diverge from this example. You might adjust thresholds, evaluation frequencies, or lookback periods based on your unique needs. Additionally, consider creating additional alerts or tailoring alert rule configurations for distinct applications and environments.

Alert rule name	Metric (signal)	Alert logic	When to evaluate	Suggested severity
High user CPU usage	CPU percentage	Threshold: `Static` Aggregation: `Average` Operator: `Greater than` Threshold value: `90`	Check every: `1 minute` Lookback period: `10 minutes`	2 - Warning
High total CPU usage	SQL instance CPU percent	Threshold: `Static` Aggregation: `Average` Operator: `Greater than` Threshold value: `90`	Check every: `1 minute` Lookback period: `10 minutes`	2 - Warning
High worker usage	Workers percentage	Threshold: `Static` Aggregation: `Minimum` Operator: `Greater than` Threshold value: `60`	Check every: `1 minute` Lookback period: `5 minutes`	1 - Error
High data IO usage	Data IO percentage	Threshold: `Static` Aggregation: `Average` Operator: `Greater than` Threshold value: `90`	Check every: `1 minute` Lookback period: `15 minutes`	3 - Informatio
Low data space	Data space used percent	Threshold: `Static` Aggregation: `Minimum` Operator: `Greater than` Threshold value: `95`	Check every: `15 minute` Lookback period: `15 minutes`	1 - Error
Low `tempdb` log space	Tempdb Percent Log Used	Threshold: `Static` Aggregation: `Minimum` Operator: `Greater than` Threshold value: `60`	Check every: `1 minute` Lookback period: `5 minutes`	1 - Error
Deadlocks	Deadlocks	Threshold: `Dynamic` Aggregation: `Total` Operator: `Greater than` Threshold sensitivity: `Medium`	Check every: `15 minutes` Lookback period: `1 hour`	3 - Informatio
Failed connections (user errors)	Failed Connections : User Errors	Threshold: `Dynamic` Aggregation: `Total` Operator: `Greater than` Threshold sensitivity: `Medium`	Check every: `5 minutes` Lookback period: `15 minutes`	2 - Warning

Figure 8.10: Alter Rule Configuration

Source: https://learn.microsoft.com/en-us/azure/azure-sql/database/monitoring-metrics-alerts?view=azuresql-db

Alert rules that rely on dynamic thresholds are designed to identify unusual metric patterns that warrant attention. These rules remain inactive until

enough historical data has been gathered to establish typical patterns.

Azure SQL Database Auditing

Azure SQL Database Auditing is a feature that helps you maintain compliance, monitor database activities, and investigate security incidents. It enables the capture of audit logs for various database events, providing a detailed record of user and application activities. Auditing can be enabled for Azure SQL Database through the Azure Portal, PowerShell, or Azure CLI. Auditing settings include the choice of storage account for audit logs and the events to be audited. Azure SQL Database Auditing allows you to audit various events, including:

Successful and failed logins: Capture information about who is accessing the database.

Query executions: Record details of queries executed, helping in troubleshooting and performance analysis.

Database modifications: Track changes to database schema, tables, and data.

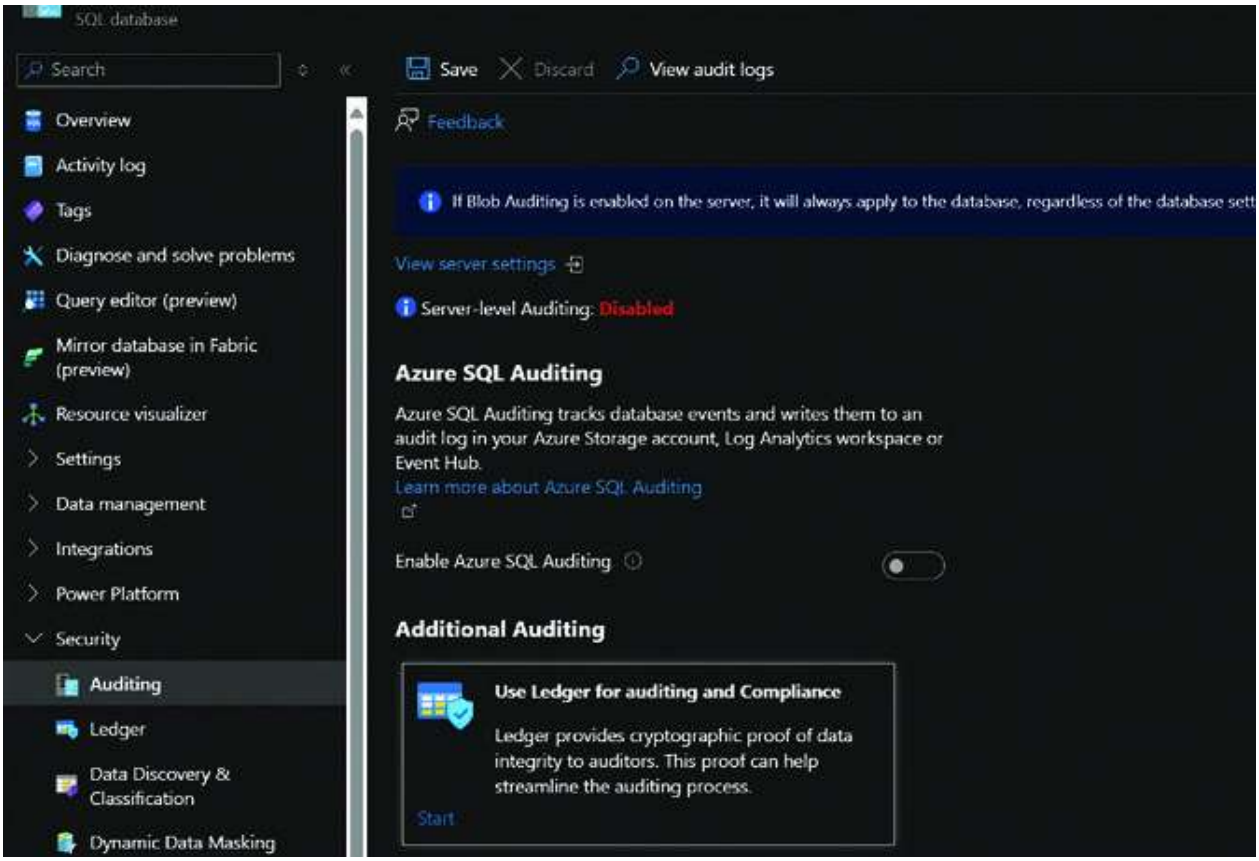


Figure 8.11: Azure SQL Auditing

Audited events are stored in Azure Storage (Azure Blob Storage or Azure Table Storage), and users need to specify a storage account when configuring auditing. Also, there is a need to define the retention period for audit logs, specifying how long logs should be retained in storage. This helps in meeting compliance requirements and maintaining historical records.

Azure SQL Database Auditing allows filtering audited events based on specific criteria. A user can choose to audit events for specific users, IP addresses, or specific types of SQL commands. Once auditing is enabled, audit logs can be queried to retrieve information about specific events. Azure Monitor Logs, Azure Storage Explorer, or other tools compatible with Azure Storage could be used to store the logs and directly query them.

One of the most important features of auditing is setting up alerts based on specific audit events, which can be notified when suspicious activities occur. For example, receive an alert when a user attempts unauthorized access or when critical data modifications are detected.

Let us go through some of the scenarios to understand this:

Scenario 1: Unauthorized Access Attempt

Event: A user attempts to log in with invalid credentials.

Audited Event: A failed login event is captured.

Action: An alert is triggered, notifying administrators of the unauthorized access attempt.

Result: The security team investigates and takes necessary actions to secure the database.

Scenario 2: Data Modification Anomaly

Event: Unexpected changes are detected in a critical database table.

Audited Event: A database modification event is captured.

Action: An alert is triggered, signaling a potential security incident.

Result: The audit logs provide details about the user and query responsible for the changes, aiding in the investigation and remediation process.

Scenario 3: Compliance Reporting

Event: Periodic compliance reporting is required for auditing purposes.

Audited Event: Various audited events, including logins, queries, and modifications.

Action: Compliance reports are generated from audit logs.

Result: The organization can demonstrate adherence to regulatory requirements by providing detailed reports on database activities.

Azure SQL Database Auditing is integrated with Azure Security Center, providing additional security insights and recommendations.

In summary, Azure SQL Database Auditing is a crucial feature for maintaining security, compliance, and accountability. It provides detailed insights into database activities, empowering organizations to detect and respond to security incidents and ensuring that they meet regulatory requirements.

The Subtle Art of Troubleshooting: SQL Server

Although we hope for a flawless software world, the reality of our current environment includes errors, exceptions, and assertions. Learning how to handle these is essential.

Therefore, it is important to acquire the skill of troubleshooting issues to ensure that you can step in and resolve those critical issues when necessary. In this section, we will explore several common troubleshooting techniques for various scenarios related to SQL Server.

Installation Issues

You may encounter occasionally installation and configuration issues when setting up SQL Server in on-premises. However, in cloud environments such as Azure, SQL is typically provided as a service—allowing you to use it like any other database without the need for manual management. For instance, Azure SQL Database exemplifies this seamless experience.

However, there is one exception: Infrastructure as a Service (IaaS) scenarios, where you deploy SQL Server on the Virtual Machines that you manage. You can choose to install SQL Server manually by deploying the VM first and then installing SQL Server afterward. In this scenario, you may sometimes notice the installation and configuration issues.

To avoid such challenges, consider deploying pre-installed SQL Server images available in the Azure Marketplace. These images simplify the setup process. Here are a few examples of pre-installed SQL Server images currently available on the Azure Marketplace.

SQL Server 2022 on Windows Server
https://azuremarketplace.microsoft.com/en-us/marketplace/apps/microsoftsqlserver.sql2022-ws2022?tab=Overview

SQL 2022 on Ubuntu Pro 20.04 LTS with 24x7
https://azuremarketplace.microsoft.com/en-us/marketplace/apps/microsoftsqlserver.sql2022-ubuntupro2004?tab=Overview

SQL Server 2022 on RHEL https://azuremarketplace.microsoft.com/en-us/marketplace/apps/microsoftsqlserver.sql2022-rhel8?tab=Overview

SQL Server 2022 on SUSE Linux Enterprise Server https://azuremarketplace.microsoft.com/en-us/marketplace/apps/microsoftsqlserver.sql2022-sles15?tab=Overview

Remember that the availability of these images may vary over time, so always check the Azure Marketplace for up-to-date options!

To address installation issues, it is essential to know where the installation log files are located. These logs provide insights into the root causes of failures. In my experience, installation issues often stem from environmental factors. Common culprits include insufficient server permissions for installation, attempting installation on a machine with a previous failed install, or running an outdated system lacking the latest patches.

By analyzing installation logs, you can pinpoint the exact cause of failure, avoiding a trial-and-error approach to issue resolution. Keep in mind that the location and installation process for these log files vary based on the operating system where SQL Server is installed.

For SQL Server on Windows, the default location for installation logs is "%programfiles%\Microsoft SQL Server\nn\Setup Bootstrap\Log." Here, 'nn' refers to the timestamped directory.

In this directory, you will find several files and subdirectories, as depicted in Figure Each of these directories is created when you attempt to run SQL Server installation, perform an upgrade, or uninstall the software. Notice that

these directories are organized based on timestamps, allowing you to refer to the one that most closely aligns with the time of your installation attempt.

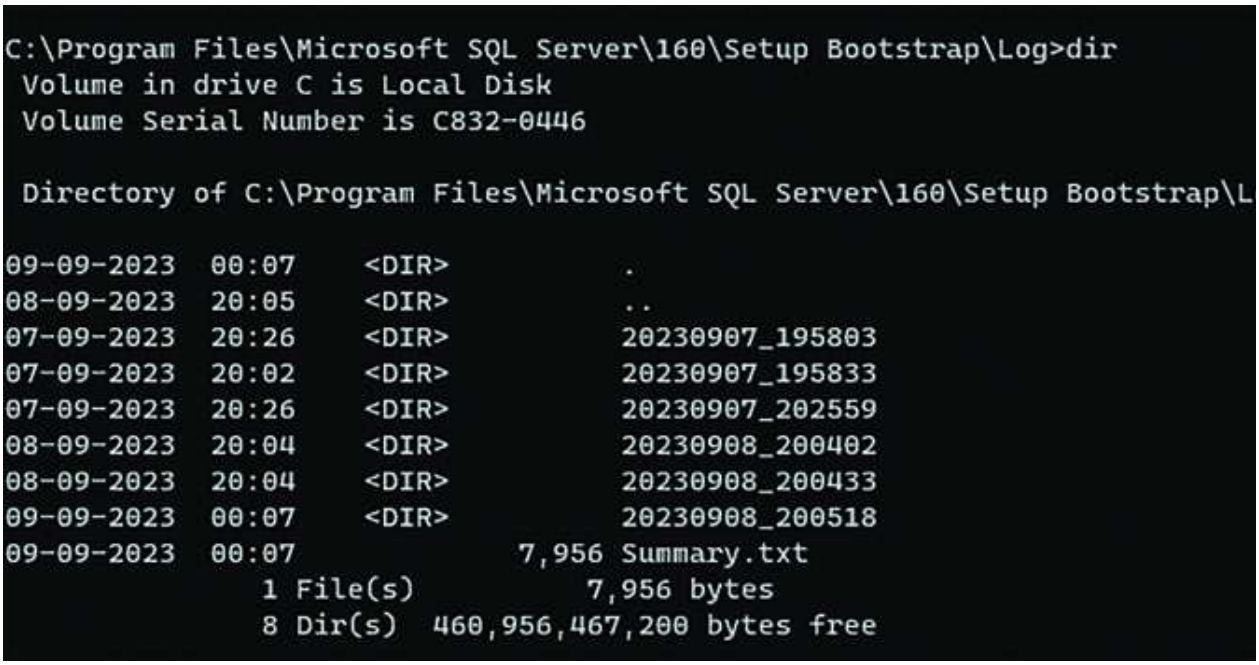


Figure 8.12: The installation logs as seen for SQL Server installed on Windows

Within these timestamped folders, you will encounter several files. However, the ones critical for identifying issues are the "Summary_Servername_timestamp.txt" files. This file provides a concise summary of the action taken, machine details, and the various user input settings provided during setup. You can also cross-reference these settings with the configuration.ini file. Additionally, at the end of the Summary file, you will find an overview helping you understand if it was a success or failure of the entire operation.

Based on the information gleaned from the summary, you can then delve into the "details.txt" file. This log captures execution details in chronological order, allowing you to understand precisely which actions failed. To further

investigate, consider reviewing these files alongside the application and event-based system logs available in Windows. You can access these logs using the Event Viewer, as depicted in Figure

For more information on these logs and other installation logs on Windows, please visit this official Microsoft documentation: https://learn.microsoft.com/en-us/sql/database-engine/install-windows/view-and-read-sql-server-setup-log-files?view=sql-server-ver16

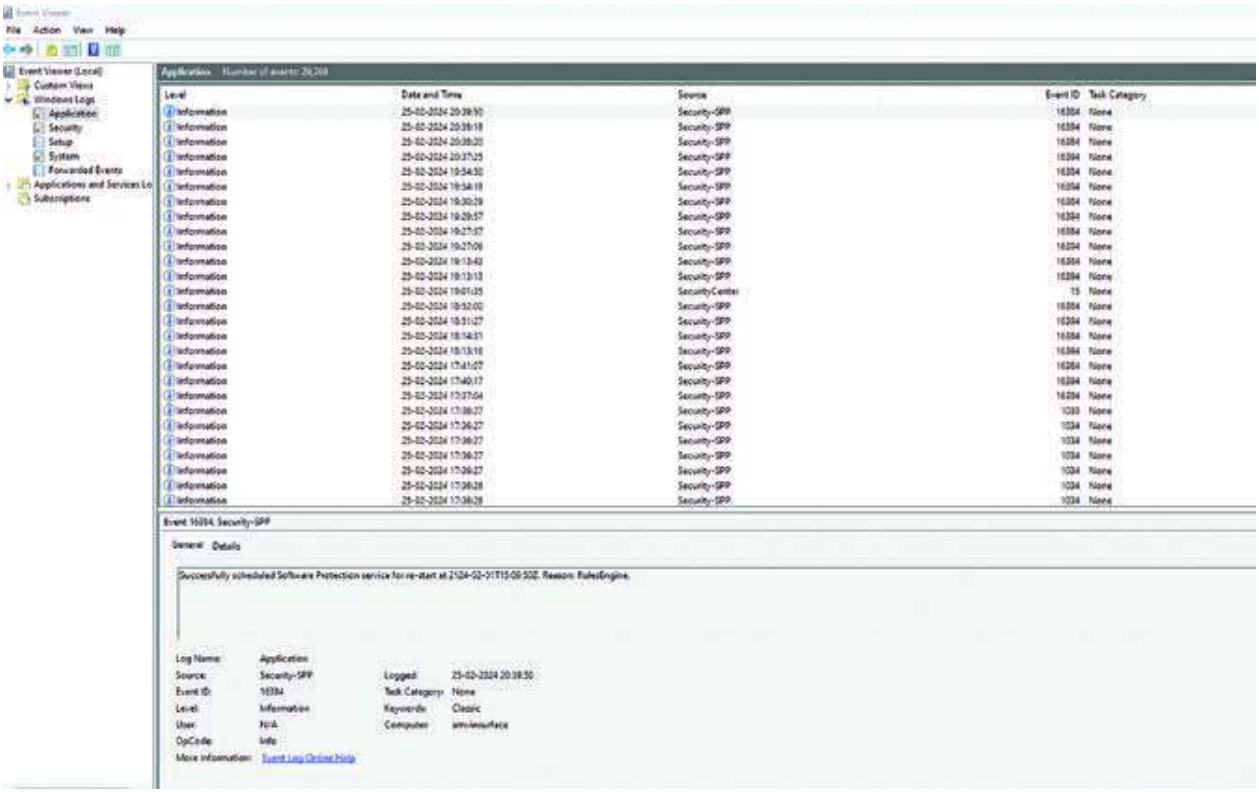


Figure 8.13: Event-based application logs as seen in Event Viewer on Windows

When installing SQL Server on Linux or container-based environments, you would not find installation logs like the familiar “summary” and “detail.txt.” Instead, these environments rely on package-based logging. Let us explore where you can locate these logs:

Red Hat (RHEL) Images:

Look in the directory “/var/log.”

Specifically, you will find installation logs at “/var/log/dnf.log.”

SUSE (SLES) Distributions:

Navigate to “/var/log.”

There, you will discover the installation logs at “/var/log/zypper.log.”

Canonical (Ubuntu) Distributions:

Head to “/var/log.”

The installation logs reside in “/var/log/dpkg.log.”

To analyze these logs, consider cross-referencing them with the system logs available in Linux environments. These system logs are typically found in the “messages” file within “/var/log/.” When reviewing the logs, keep an eye out for the “mssql” package. You can search for mssql in these logs using the grep function.

Issues with SQL Server installation on Linux are rarely encountered, so it is likely that there will be no need to delve into these files.

Configurational Issues

For a successful SQL Server deployment, it is essential to install and configure SQL Server. Once the SQL Server-related packages are installed, the next step involves configuration. This includes tasks such as setting up startup accounts for various SQL Server services, such as the SQL Server Agent or the browser service. It also involves steps to configure tempdb, the location of the database and log files in case you change from its default location, configuring the SKU of the SQL Server instance, and more.

If you encounter configuration issues and need to troubleshoot, consider checking the datastore folder located at: %programfiles%\Microsoft SQL Server\nn\Setup Bootstrap\Log. This folder contains logs for all configuration steps organized by the package being configured. It proves invaluable for diagnosing configuration errors.

For SQL Server on Linux and containers, configuration is handled via the command: sudo /opt/mssql/bin/mssql-conf setup. Execute this command immediately after completing the SQL Server installation. It allows you to configure the edition (SKU), set the ‘sa’ account password, and accept or decline any displayed End User License Agreements (EULAs).

To customize and modify configurations for your SQL Server deployment, you have several options. To view a comprehensive list of these options, execute the following command:

“sudo /opt/mssql/bin/mssql-conf –help”. You can find an example of this in Figure

Figure 8.14: Various configuration options available using the mssql-conf for SQL Server on Linux deployments

The configuration log “mssql-conf.log” for SQL Server on Linux-based distributions is created during a failure at the “/var/opt/mssql/log/mssql-conf/.” Normally the configuration scripts that are run to configure SQL Server are located at: “/opt/mssql/lib/mssql-conf,” and when the failure occurs, the command prompt displays what the issue is, as shown in the Figure where the checkinstall.sh was missing.

```
[amvin@sqlrhel9 mssql-conf]$ sudo /opt/mssql/bin/mssql-conf setup
Warning: could not create log file for mssql-conf at /var/opt/mssql/log/mssql-conf/mssql-conf.log.
Traceback (most recent call last):
  File "/opt/mssql/bin/../lib/mssql-conf/mssql-conf.py", line 600, in <module>
    main()
  File "/opt/mssql/bin/../lib/mssql-conf/mssql-conf.py", line 596, in main
    processCommands()
  File "/opt/mssql/bin/../lib/mssql-conf/mssql-conf.py", line 312, in processCommands
    COMMAND_TABLE[args.which]()
  File "/opt/mssql/bin/../lib/mssql-conf/mssql-conf.py", line 95, in handleSetup
    mssqlconfhelper.setupSqlServer(eulaAccepted, noprompt=args.noprompt)
  File "/opt/mssql/lib/mssql-conf/mssqlconfhelper.py", line 1073, in setupSqlServer
    if not checkInstall():
  File "/opt/mssql/lib/mssql-conf/mssqlconfhelper.py", line 1043, in checkInstall
    return runScript(checkInstallScript, runAsRoot) == 0
  File "/opt/mssql/lib/mssql-conf/mssqlconfhelper.py", line 1024, in runScript
    process = subprocess.run([pathToScript], stdout=subprocess.PIPE, stderr=subprocess.STDOUT)
  File "/usr/lib64/python3.9/subprocess.py", line 505, in run
    with Popen(*popenargs, **kwargs) as process:
  File "/usr/lib64/python3.9/subprocess.py", line 951, in __init__
    self._execute_child(args, executable, preexec_fn, close_fds,
  File "/usr/lib64/python3.9/subprocess.py", line 1821, in _execute_child
    raise child_exception_type(errno_num, err_msg, err_filename)
FileNotFoundError: [Errno 2] No such file or directory: '/opt/mssql/lib/mssql-conf/checkinstall.sh'
```

Figure 8.15: A sample example of a configuration error for SQL Server on Linux deployment

General Troubleshooting

In the realm of SQL Server, various scenarios and features can occasionally lead to issues. In this section, we will explore general troubleshooting techniques that can assist you in these situations. Drawing from experience, here is a list of the most common SQL Server-related issues.

Troubleshooting connectivity issues, including login failures and connection timeouts.

Handling tasks related to high availability, disaster recovery, replication, SQL Server multi-server agents, and other features of SQL Server.

Addressing Active Directory (AD) authentication issues in SQL Server on Linux environments.

When encountering any of these scenarios, it is recommended to perform the following steps to effectively troubleshoot and resolve the issue:

Begin with the Error Pay attention to the error message you receive. For instance, in the case of a login failure issue, the message might resemble what is shown in Figure

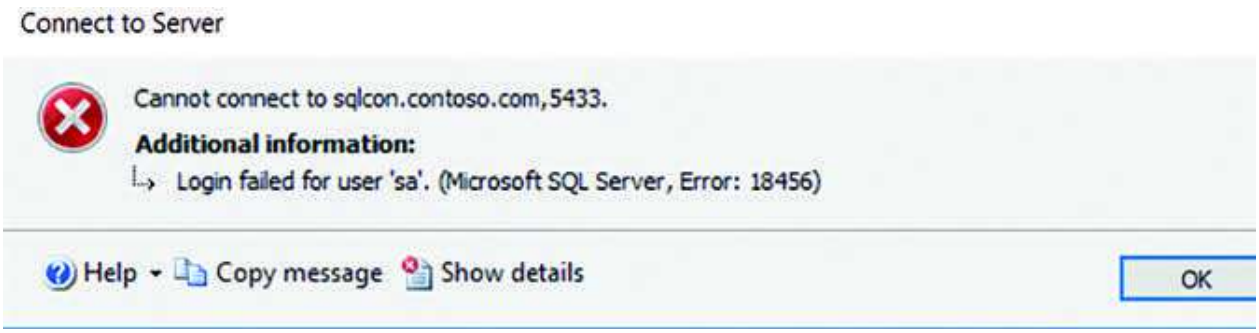


Figure 8.16: Login failed a generic error message

While understanding the general issue is essential, it does not always reveal the specific cause behind a login failure message. Sometimes, for security reasons, the complete error message and its underlying reason are intentionally concealed. Consider a scenario where an unauthorized attempt is made to log in maliciously to your SQL Server instance. If the error message explicitly stated that the login name ‘sa’ does not exist or that the password is incorrect, it would inadvertently guide the malicious user to correct their previous attempt and retry. By keeping error messages generic, an additional layer of security is maintained.

Keep in mind that not all error messages are generic. While some specific messages fall into the generic category, others, as depicted in Figure provide precise details about the issue, and guide you through the necessary steps for resolution.

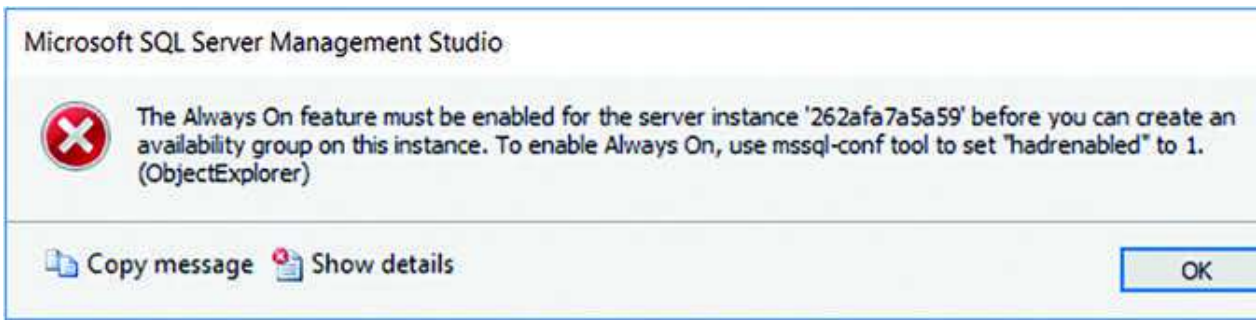


Figure 8.17: Error message with exact issue and resolution step displayed in SQL Server

Cross-reference the issue with the SQL Server Errorlogs: Take note of the timestamp when the issue occurred and cross-reference it with the SQL Server Error logs. These logs capture most of the operations performed by SQL Server and can provide essential details for troubleshooting.

For SQL Server on Windows, the error log file (including errorlog.n files) is typically located at: %Program Files%\Microsoft SQL Server\MSSQL.n\MSSQL\LOG\, with the errorlog file being the most recent.

For SQL Server on Linux, the error log files reside in: /var/opt/mssql/log."

In the example of the login failure, when we examine the error log and observe the timestamp corresponding to the login failure message, we encounter the following details. These specifics precisely indicate the reason for the login failure, enabling us to take corrective actions and resolve the issue effectively.

2024-02-26 13:19:59.920 Logon Error: 18456, Severity: 14, State: 8.

2024-02-26 13:19:59.920 Logon Login failed for user 'sa'. Reason: Password did not match that for the login provided. [CLIENT: 10.0.0.4]

Query and relate the information from the System catalog views, the Dynamic Management views (DMVs), and the Dynamic Management Functions (DMFs): If you are unable to find relevant information from error messages and logs, the next step is to explore Dynamic Management Views

(DMVs) and system catalog views. These are particularly useful when troubleshooting specific features in SQL Server. For instance: Always On Availability Groups: If you encounter issues such as synchronization problems, DMVs can provide insights. For Transactional Log Management: Troubleshooting log-related problems benefits from querying relevant DMVs. For In-Memory or Columnstore Index Performance: To compare row counts in rowgroups versus columnstore, DMVs are invaluable.

Refer to the official Microsoft documentation for a comprehensive list. Then, query the specific DMVs related to the feature you are troubleshooting.

Finally, query the system logs and collect If none of the preceding steps lead to a resolution. Then you will need to correlate and cross-reference multiple logs, focusing on the common link: the timestamp of the issue. Here is what you can do:

System Refer to the Event Viewer logs in Windows and the messages log for Linux distributions. These logs often hold valuable clues.

Collect Additional While reproducing the issue, gather data such as Xevents and Profiler Traces. Review this collected data to understand why the issue occurs.

Comparing Successful versus Error Comparing data collected during a successful event with data from an error during the same event helps pinpoint the exact problem area.

Remember, persistence and thorough analysis are key when troubleshooting complex issue.

Troubleshooting Performance-Related issues

If you have ever worked as a database administrator, developer, or database engineer, you have likely encountered these familiar statements:

“My queries are running slow.”

“The server is overall very sluggish.”

“All my queries are taking long to complete.”

These questions often arise when investigating database performance issues. Whether you have posed them yourself or received them from others, addressing slow queries is a common task in the realm of database maintenance.

In this section, we will explore effective strategies for addressing performance problems encountered in SQL Server. To be candid, resolving performance issues within SQL Server is easy. Thanks to recent enhancements such as Intelligent Query Processing (IQP), identifying and resolving these issues has become relatively simple. Unlike previous sections where error messages guided our troubleshooting, performance issues often lack such direct cues.

Dealing with performance issues can become overwhelming if not tackled methodically. Essentially, you are examining the current state of the server or query, which may not align with your expectations. Consequently, you might

find yourself with a database server that does not execute queries as anticipated.

The flow chart shown in Figure 8.18 should help you get started with the troubleshooting of the performance issue. Let us traverse this together:

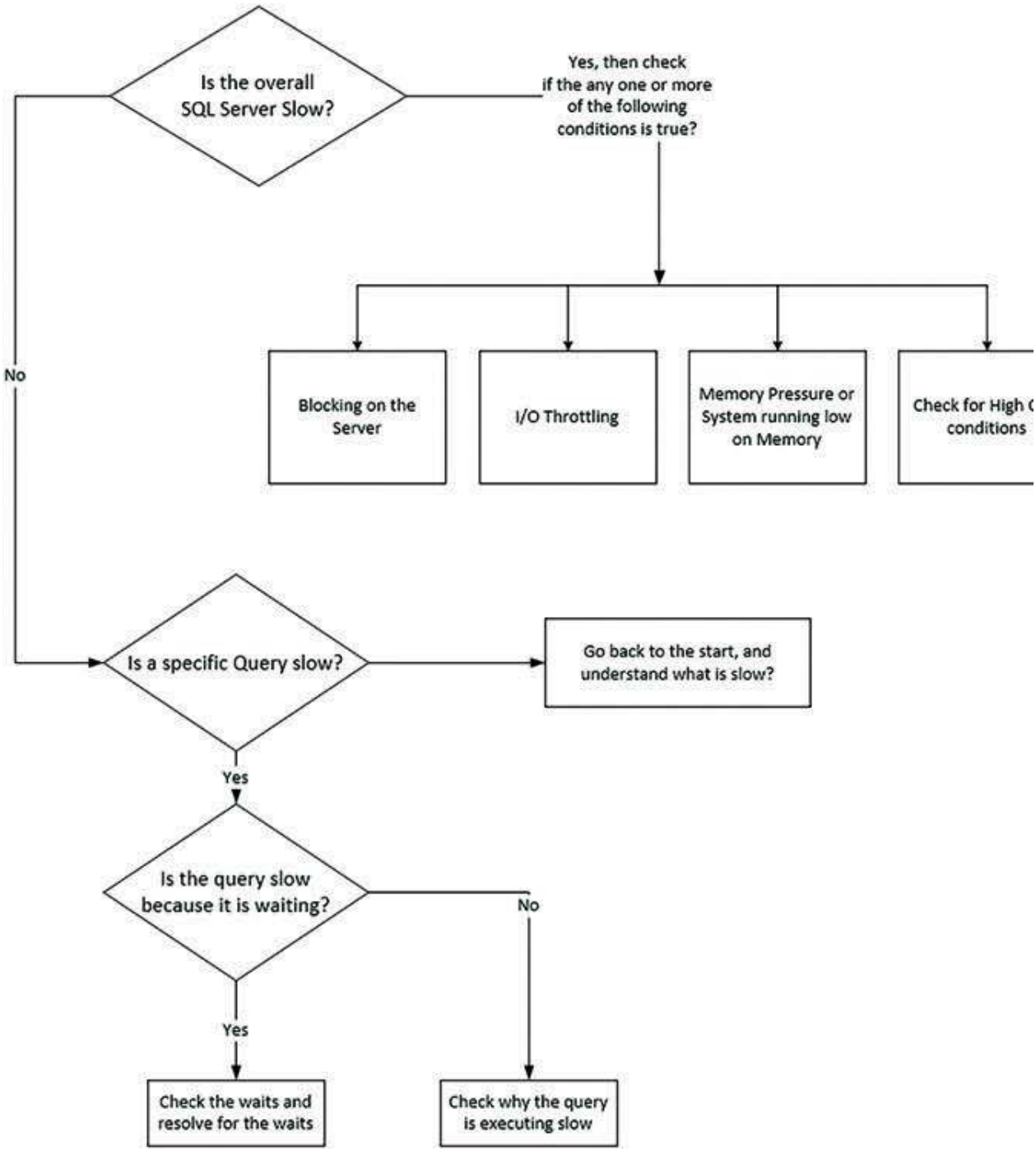


Figure 8.18: Performance troubleshooting flow chart

So, the first question that you ask yourself is:

“Is the overall SQL Server slow?”

If the answer is yes, your objective is to identify the reasons behind the sluggishness. Keep in mind that these issues can occur individually or in combination, and there may be other unlisted factors contributing to the slowness.

The goal here is to guide you on where to start looking and what data to collect.

Blocking:

In any Relational Database Management System (RDBMS) with lock-based concurrency, blocking is a natural occurrence.

When one session holds a lock on a specific resource and another session tries to acquire a conflicting lock, blocking occurs.

The critical factor is the frequency and duration of this blocking, as it can significantly impact performance.

To address this:

Collect relevant data from Dynamic Management Views (DMVs) such as sys.dm_exec_sessions, Extended Events, and profiler traces.

Identify the head blocker(s) and analyze the query or queries responsible for prolonged blocking.

I/O

To achieve optimized performance from SQL Server, ensure that the I/O storage used in the backend is capable of handling the database workload.

If you encounter I/O related wait types for multiple queries such as PAGEIOLATCH (EX/SH/UP), WRITELOG, AYSNC_IO_COMPLETION, or BACKUP_IO when reviewing the DMVs such as sys.dm_exec_sessions and this points to IO related issues.

Additionally, watch out for error 833 messages in the error log: “SQL Server has encountered XX occurrence(s) of I/O requests taking longer than 15 seconds to complete on File <%path%> in database id X.” These messages also point to I/O throttling.

Data Collection for Troubleshooting:

You can collect and review the perfmon counters, such as the disk bytes/sec, disk read bytes/sec, and disk write bytes/sec. Analyze these counters to determine whether the SQL Server process is driving high I/O or if another process on the server is causing the high I/O, with SQL Server as the victim. For Linux-based environments, you can monitor the disk IO using tools such as:

https://man7.org/linux/man-pages/man1/sar.1.html

Memory Pressure (System running on low memory):

Optimized query execution in relational database systems relies on having sufficient memory available.

When your server runs on low memory, you will notice sluggishness at the server level.

Common Indicators of Low Memory

Look out for the following signs:

Memory-related errors in the SQL Server error log, such as:

Failure to allocate sufficient memory for query execution.

Failure to allocate memory for locks.

Failure to allocate pages in the buffer pool, and more.

Occurrence of SQL Server out of memory (OOM) dumps.

Memory-related waits, including:

MEMORY_ALLOCATION_EXT

RESOURCE_SEMAPHORE_*

To determine whether memory issues are external (caused by other applications consuming memory on the same server) or internal (due to SQL Server memory management),

Collect and review perfmon counters such as:

Available MBytes

Working Set

Private Bytes

SQL Manager: (all counters)

SQL Manager: (all counters)

Refer the memory related DMVs and review the DBCC MemoryStatus output that provides the overall memory usage by different memory managers.

High CPU condition:

The CPU is a critical server resource that can impact overall performance.

High CPU usage is noticeable when:

Task Manager consistently shows CPU utilization above 100%.

Or on the SQL Server side, you observe high waits for many queries with the wait type SOS_SCHEDULER_YIELD.

To address this:

Determine whether the CPU is consumed by SQL Server or other applications on the server.

Collect and review perfmon counters such as:

Process/%User Time

Process/% Privileged Time

If the SQL Server process is indeed consuming the CPU, investigate further using DMVs such as sys.dm_os_schedulers and sys.dm_exec_requests

If the overall server is not slow but a specific query is sluggish, you must determine whether query is a “waiter” or a “runner.” Here is how to analyze it:

Examine the following DMVs:

sys.dm_exec_requests (for currently running queries)

sys.dm_exec_query_stats (for past query executions)

Focus on three specific columns:

Elapsed time (ms)

CPU time (ms)

Reads (logical)

Interpret the results:

If the CPU time is close to or higher than the elapsed time, consider the query a “runner.”

If the elapsed time significantly exceeds the CPU time, it qualifies as a “waiter” query.

If a query is identified as a “waiter” (as depicted in the flowchart in Figure the next step is to examine the wait types associated with it and address them accordingly.

If a query is categorized as a “runner” but still exhibits slowness, there are two potential reasons to consider:

Long Compilation Time:

When executing a query in SQL Server, it undergoes a compilation stage. During compilation:

The query is parsed to identify all the necessary objects involved.

Alternative execution plans are considered.

The most optimized plan is selected for query execution.

If the query takes an extended time to compile, SQL Server evaluates multiple execution plans before settling on the one it deems most efficient.

To identify lengthy compilation times:

Examine the XML plan, which reveals details such as Compile Time and Compile CPU.

If you can execute the query at will, enable set statistics time on and observe the parse and compile time metrics (as shown in Figure

```
set statistics time on
--set statistics IO on
SELECT t.text,
        qs.total_elapsed_time / qs.execution_count
        AS avg_elapsed_time,
        qs.total_worker_time / qs.execution_count
        AS avg_cpu_time,
        (qs.total_elapsed_time - qs.total_worker_time) / qs.execution_count
        AS avg_wait_time,
        qs.total_logical_reads / qs.execution_count
        AS avg_logical_reads,
        qs.total_logical_writes / qs.execution_count
        AS avg_writes,
        qs.total_elapsed_time
        AS cumulative_elapsed_time
FROM sys.dm_exec_query_stats qs
        CROSS apply sys.Dm_exec_sql_text (sql_handle) t
WHERE (qs.total_elapsed_time - qs.total_worker_time) / qs.total_elapsed_time
        > 0.2
ORDER BY qs.total_elapsed_time / qs.execution_count DESC
```

```
%
Results  Messages
SQL Server parse and compile time:
   CPU time = 0 ms, elapsed time = 0 ms.

(0 rows affected)

 SQL Server Execution Times:
   CPU time = 0 ms,  elapsed time = 3 ms.

Completion time: 2024-02-28T02:35:27.8865058+05:30
```

Figure 8.19: Enable Statistics time on to see the compile versus execution time of the query when executed

To address long compilation times for queries, consider the following steps:

Keep SQL Server Updated:

Ensure that you are running the latest updates for SQL Server to avoid known issues related to compilation.

Optimize Query Structure:

Long compilations often occur in queries with many IN or OR clauses.

Consider using tables and joins instead of complex IN or OR conditions.

Aim to keep the number of joins in a single query below 100, as excessive joins can contribute to extended compile times.

Update Statistics:

Regularly update statistics for the objects involved in the query.

Outdated statistics can lead SQL Server to choose suboptimal execution plans.

Force Parameterization:

Explore using the force parameterization option for such queries.

This helps avoid unnecessary compilations.

Long Execution Queries:

If query execution consumes the most time, consider reviewing the execution plan to identify the specific operator or set of operators responsible for the delay. Here are the steps:

Review Execution Plan:

Examine the execution plan to pinpoint the operators causing the most significant time overhead.

Identify bottlenecks, such as expensive joins, sorts, or scans.

Index Optimization:

Focus on the objects (tables or views) associated with the time-consuming operator(s).

Consider creating indexes on these objects to improve query performance.

Indexes can enhance data retrieval efficiency and reduce execution time.

Modify Query or Provide Statistics:

Based on the execution plan analysis:

Modify the query if possible (example, rewrite complex joins, optimize subqueries).

Ensure that statistics are up-to-date for accurate cardinality estimates.

Provide enough information for SQL Server to choose an optimized query plan.

As we have explored various scenarios and troubleshooting approaches, here is a convenient list of tools specifically useful for collecting and analyzing data during performance-related investigations.

PSSDIAG: One of the powerful tools that helps you collect data during the time of the issue.

Data Sources: It natively collects data from:

Performance Monitor logs

SQL Profiler traces

SQL Server blocking script output

Windows Event Logs

SQLDIAG output

You can enable or disable specific log types and adjust sample intervals.

Platform Compatibility: PSSDIAG works for both SQL Server on Windows and Linux environments.

SQL Log A Comprehensive Data Collection Tool

SQL Log Scout is a versatile data collection utility designed for SQL Server systems.

It allows you to collect diagnostic logs based on specific scenarios. Supported scenarios include:

GeneralPerf

DetailedPerf

Replication

AlwaysOn

NetworkTrace

Memory, and more

Customized Data Collection:

Depending on the issue you are facing, configure SQL Log Scout for the relevant scenario.

Collect data from various sources:

Operating System (OS): Gather OS-related information.

Performance Monitor (perfmon) counters: Monitor system performance metrics.

SQL Server DMVs: Extract insights from dynamic management views.

SQL Log Scout simplifies data collection, aiding in troubleshooting and performance analysis.

SQLNexus: Up until now, we have explored various data collection tools. SQL Nexus stands out as a powerful tool for pinpointing the root causes of SQL Server performance issues. Here's why:

Data Loading and

SQL Nexus allows you to load collected data from various sources, such as pssdiag and SQL LogScout.

It provides robust analysis capabilities to dissect the data and identify performance bottlenecks.

Graphical

SQL Nexus goes beyond raw data.

It generates graphs and visualizations for quick reference and deeper analysis.

These visual representations help you understand trends, anomalies, and critical areas affecting performance.

Conclusion

In conclusion, this chapter has delved into the intricate world of monitoring, performance tuning, and optimization within SQL Server, emphasizing its application across diverse platforms and environments, with a particular focus on Azure SQL Database. From Windows to Linux, containers, and Azure SQL, the chapter has offered an array of invaluable insights, ranging from essential tips and tricks to advanced techniques. By introducing a sample monitoring solution leveraging Telegraf, InfluxDB, and Grafana, readers are equipped with practical tools to effectively monitor SQL Server instances. Furthermore, the chapter navigates through the labyrinth of error handling, showcasing how errors, exceptions, and assertions can be managed adeptly using error messages, logs, and DMVs.

In the realm of Azure SQL Database, the chapter illuminates the groundbreaking features of automatic tuning and performance recommendations, powered by artificial intelligence and machine learning. Through illuminating use cases, readers gain a deeper understanding of how these cutting-edge technologies optimize database performance and fortify security measures. Moreover, the chapter offers a comprehensive guide on leveraging query performance insight and query store to identify and optimize poorly performing queries, coupled with insights on Azure monitor and alerting for seamless dashboard creation and anomaly detection.

Lastly, the chapter underscores the importance of auditing in Azure SQL Database, shedding light on its pivotal role in ensuring compliance, monitoring database activities, and responding effectively to security incidents. Armed with these insights and techniques, readers are empowered to navigate the complex landscape of SQL Server with confidence and proficiency.

Additional Resources

Get Started Grafana-

InfluxDB -

Importing a Dashboard -

Monitor Overview -

Query Performance Insight -

High CPU diagnose -

Database advisor

Monitoring Metrics -

SQL Server 2022 on Windows Server 2022 -

SQL 2022 on Ubuntu Pro 20.04 LTS with 24x7 Support -

SQL Server 2022 on RHEL HA-8.6 -

SQL Server 2022 on SUSE Linux Enterprise Server 15 -

View and Read SQL Server Setup logs

System Dynamic Management Views -

Extended Events -

iostat -

iotop -

sar -

A

B

C

D

E

F

H

I

J

K

L

M

P

R

S

T

W