

2ND EDITION

Azure Data Engineering Cookbook

Get well versed in various data engineering techniques
in Azure using this recipe-based guide



NAGARAJ VENKATESAN | AHMAD OSAMA

Azure Data Engineering Cookbook

Second Edition

Get well versed in various data engineering techniques in Azure using this recipe-based guide

Nagaraj Venkatesan

Ahmad Osama



BIRMINGHAM—MUMBAI

Azure Data Engineering Cookbook

Second Edition

Copyright © 2022 Packt Publishing

All rights reserved. No part of this book may be reproduced, stored in a retrieval system, or transmitted in any form or by any means, without the prior written permission of the publisher, except in the case of brief quotations embedded in critical articles or reviews.

Every effort has been made in the preparation of this book to ensure the accuracy of the information presented. However, the information contained in this book is sold without warranty, either express or implied. Neither the authors, nor Packt Publishing or its dealers and distributors, will be held liable for any damages caused or alleged to have been caused directly or indirectly by this book.

Packt Publishing has endeavored to provide trademark information about all of the companies and products mentioned in this book by the appropriate use of capitals. However, Packt Publishing cannot guarantee the accuracy of this information.

Publishing Product Manager: Reshma Raman

Senior Editor: Nazia Shaikh

Content Development Editor: Manikandan Kurup

Technical Editor: Sweety Pagaria

Copy Editor: Safis Editing

Project Coordinator: Farheen Fathima

Proofreader: Safis Editing

Indexer: Sejal Dsilva

Production Designer: Joshua Misquitta

Marketing Coordinators: Priyanka Mhatre and Nivedita Singh

First published: March 2021

Second edition: September 2022

Production reference: 2070922

Published by Packt Publishing Ltd.

Livery Place

35 Livery Street

Birmingham

B3 2PB, UK.

ISBN 978-1-80324-678-9

www.packtpub.com

Contributors

About the authors

Nagaraj Venkatesan works as a cloud solution architect at Microsoft. At Microsoft, he works with some of the largest companies in the world, solving their complex data engineering problems and helping them build effective solutions using cutting-edge technologies based on Azure. Nagaraj, based out of Singapore, is a popular member of the data and AI community and is a regular speaker at several international data and AI conferences. He is a two-time **Microsoft Most Valuable Professional (MVP)** award winner, in 2016 and 2017. Nagaraj shares his technical expertise through his blog and on his YouTube channel called *DataChannel*. He also holds a master's degree in computing from the National University of Singapore.

Ahmad Osama works for Pitney Bowes Pvt. Ltd. as a technical architect and is a former Microsoft Data Platform MVP. In his day job, he works on developing and maintaining high performant, on-premises and cloud SQL Server OLTP environments as well as deployment and automating tasks using PowerShell. When not working, Ahmad blogs at DataPlatformLabs and can be found glued to his Xbox.

About the reviewers

Milind Kumar Chaudhari is an experienced cloud data engineer/architect who specializes in designing, implementing, and managing complex data pipelines. He is a data enthusiast and passionate about leveraging the best data engineering practices to solve critical business problems. He is also currently a technical reviewer with O'Reilly Media.

Vijay Kumar Suda is a senior cloud big data architect with over 20 years of experience working with global clients. He has worked in Switzerland, Belgium, Mexico, Bahrain, India, and Canada and helped customers across multiple industries. He has been based out of the USA since 2008. His expertise includes data engineering, architecture, the cloud, AI, and machine learning.

Firstly, I'd like to thank my parents, Sri Koteswara Rao and Rajyalakshmi, for their love and support in every step of my life. I'd like to thank my wife, Radhika, my son, Chandra, and my daughter, Akshaya, for their daily support and patience. I'd like to thank my siblings, Rama, Swarana, and Dr. SriKumar, for their support, and finally, I'd like to thank Packt for the opportunity to review this book.

Table of Contents

Preface	xv
---------	----

1

Creating and Managing Data in Azure Data Lake	1
---	---

Technical requirements	2	Getting ready	8
Provisioning an Azure storage account using the Azure portal	2	How to do it...	8
Getting ready	2	How it works...	10
How to do it...	3	Managing blobs in Azure	
How it works...	6	Storage using PowerShell	10
Provisioning an Azure storage account using PowerShell	6	Getting ready	10
How to do it...	6	How to do it...	11
How it works...	6	How it works...	16
Creating containers and uploading files to Azure Blob storage using PowerShell	8	Configuring blob lifecycle management for blob objects using the Azure portal	16
Getting ready	6	Getting ready	17
How to do it...	7	How to do it...	17
How it works...	7	How it works...	20

2

Securing and Monitoring Data in Azure Data Lake	21
---	----

Configuring a firewall for an Azure Data Lake account using the Azure portal	22	How to do it...	22
Getting ready	22	How it works...	24
		Configuring virtual networks for an Azure Data Lake account using the Azure portal	24

Getting ready	24	Accessing Blob storage accounts using managed identities	39
How to do it...	24	Getting ready	40
How it works...	27	How to do it...	40
Configuring private links for an Azure Data Lake account	28	How it works...	45
Getting ready	28	Creating an alert to monitor an Azure storage account	45
How to do it...	28	Getting ready	45
How it works...	31	How to do it...	46
Configuring encryption using Azure Key Vault for Azure Data Lake	34	How it works...	54
Getting ready	34	Securing an Azure storage account with SAS using PowerShell	54
How to do it...	35	Getting ready	54
How it works...	38	How to do it...	55
		How it works...	58

3

Building Data Ingestion Pipelines Using Azure Data Factory	59		
Technical requirements	60	Triggering a pipeline in Azure Data Factory	81
Provisioning Azure Data Factory	60	Getting ready	82
How to do it...	60	How to do it...	82
How it works...	62	How it works...	86
Copying files to a database from a data lake using a control flow and copy activity	62	Copying data from a SQL Server virtual machine to a data lake using the Copy data wizard	87
Getting ready	62	Getting ready	87
How to do it...	64	How to do it...	91
How it works...	78	How it works...	96

4**Azure Data Factory Integration Runtime** **99**

Technical requirements	100	How to do it...	120
Configuring a self-hosted IR	100	How it works...	128
Getting ready	100	Patching a self-hosted IR	128
How to do it...	100		
How it works...	113	Getting ready	128
Configuring a shared self-hosted IR	113	How to do it...	129
How to do it...	113	How it works...	135
Getting ready	113	Migrating an SSIS package to Azure Data Factory	135
How to do it...	114		
Configuring high availability for a self-hosted IR	120	Getting ready	135
How to do it...	120	How to do it...	136
Getting ready	120	How it works...	148

5**Configuring and Securing Azure SQL Database** **151**

Technical requirements	152	Getting ready	163
Provisioning and connecting to an Azure SQL database using PowerShell	152	How to do it...	164
Getting ready	152	How it works...	171
How to do it...	152	Configuring Azure Key Vault for Azure SQL Database	171
How it works...	155		
Getting ready	155	Getting ready	171
How to do it...	155	How to do it...	171
How it works...	155	How it works...	176
Implementing an Azure SQL Database elastic pool using PowerShell	156	Provisioning and configuring a wake-up script for a serverless SQL database	177
Getting ready	156		
How to do it...	156	Getting ready	178
How it works...	163	How to do it...	178
How it works...	163	How it works...	192
Configuring a virtual network and private endpoints for Azure SQL Database	163		

Configuring the Hyperscale tier of Azure SQL Database	192	Getting ready	193
		How to do it...	193

6

Implementing High Availability and Monitoring in Azure SQL Database **199**

Implementing active geo-replication for an Azure SQL database using PowerShell	200	How it works...	218
Getting ready	200	Implementing vertical scaling for an Azure SQL database using PowerShell	219
How to do it...	200	Getting ready	219
How it works...	204	How to do it...	219
How it works...		How it works...	232
Implementing an auto-failover group for an Azure SQL database using PowerShell	205	Monitoring an Azure SQL database using the Azure portal	232
Getting ready	205	Getting ready	232
How to do it...	205	How to do it...	232
How it works...	210		
Configuring high availability to the Hyperscale tier of Azure SQL Database	211	Configuring auditing for Azure SQL Database	242
Getting ready	211	Getting ready	242
How to do it...	211	How to do it...	242
		How it works...	246

7

Processing Data Using Azure Databricks **247**

Technical requirements	248	Integrating Databricks with Azure Key Vault	255
Configuring the Azure Databricks environment	248	Getting ready	255
Getting ready	248	How to do it...	257
How to do it...	248	How it works...	259

Mounting an Azure Data Lake container in Databricks	260	Getting ready	282
Getting ready	260	How to do it...	282
How to do it...	261	How it works...	292
How it works...	270	Working with Delta Lake tables	292
Processing data using notebooks	271	Getting ready	292
Getting ready	271	How to do it...	292
How to do it...	271	How it works...	300
How it works...	281	Connecting a Databricks Delta Lake table to Power BI	300
Scheduling notebooks using job clusters	282	Getting ready	300
		How to do it...	300
		How it works...	310

8

Processing Data Using Azure Synapse Analytics	311		
Technical requirements	312	Getting ready	325
Provisioning an Azure Synapse Analytics workspace	312	How to do it...	325
Getting ready	312	How it works...	331
How to do it...	312	Querying the data in a lake database from serverless SQL pool	332
Analyzing data using serverless SQL pool	315	Getting ready	332
Getting ready	315	How to do it...	332
How it works...	321	How it works...	336
Provisioning and configuring Spark pools	322	Scheduling notebooks to process data incrementally	336
Getting ready	322	Getting ready	337
How to do it...	322	How to do it...	337
How it works...	325	How it works...	344
Processing data using Spark pools and a lake database	325	Visualizing data using Power BI by connecting to serverless SQL pool	345
		Getting ready	345
		How to do it...	345
		How it works...	349

9

Transforming Data Using Azure Synapse Dataflows	351		
Technical requirements	352	How it works...	378
Copying data using a Synapse data flow	352	Configuring partitions to optimize data flows	379
Getting ready	352	Getting ready	379
How to do it...	352	How to do it...	379
How it works...	364	How it works...	383
Performing data transformation using activities such as join, sort, and filter	364	Parameterizing Synapse data flows	383
Getting ready	364	Getting ready	383
How to do it...	365	How to do it...	384
How it works...	375	How it works...	388
Monitoring data flows and pipelines	375	Handling schema changes dynamically in data flows using schema drift	389
Getting ready	376	Getting ready	389
How to do it...	376	How to do it...	389
		How it works...	398

10

Building the Serving Layer in Azure Synapse SQL Pool	399		
Technical requirements	400	Getting ready	408
Loading data into dedicated SQL pools using PolyBase and T-SQL	400	How to do it...	409
Getting ready	400	How it works...	413
How to do it...	401		
How it works...	408	Creating distributed tables and modifying table distribution	413
Getting ready	408	Getting ready	413
How to do it...	408	How to do it...	414
How it works...	408	How it works...	417
Loading data into a dedicated SQL pool using COPY INTO	408		

Creating statistics and automating the update of statistics	417	Implementing workload management in an Azure Synapse dedicated SQL pool	430
Getting ready	417	Getting ready	430
How to do it...	418	How to do it...	430
How it works...	422	How it works...	435
Creating partitions and archiving data using partitioned tables	422	Creating workload groups for advanced workload management	436
Getting ready	422	Getting ready	436
How to do it...	423	How to do it...	436
How it works...	430	How it works...	442

11

Monitoring Synapse SQL and Spark Pools **443**

Technical requirements	444	Creating workbooks in a Log Analytics workspace to visualize monitoring data	459
Configuring a Log Analytics workspace for Synapse SQL pools	444	Getting ready	459
Getting ready	444	How to do it...	460
How to do it...	444	How it works...	466
How it works...	447	Monitoring table distribution, data skew, and index health using Synapse DMVs	467
Configuring a Log Analytics workspace for Synapse Spark pools	448	Getting ready	467
Getting ready	448	How to do it...	468
How to do it...	448	Building monitoring dashboards for Synapse with Azure Monitor	473
How it works...	451	Getting ready	473
Using Kusto queries to monitor SQL and Spark pools	451	How to do it...	473
Getting ready	451	How it works...	482
How to do it...	452		
How it works...	459		

12

Optimizing and Maintaining Synapse SQL and Spark Pools	483		
Technical requirements	484	Getting ready	506
Analyzing a query plan and fixing table distribution	484	How to do it...	506
Getting ready	485	How it works...	514
How to do it...	485	Auto pausing Synapse dedicated SQL pool	514
How it works...	493	Getting ready	515
How to do it...	493	How to do it...	515
How it works...	493	How it works...	522
Monitoring and rebuilding a replication table cache	493	Optimizing Delta tables in a Synapse Spark pool lake database	522
Getting ready	493	Getting ready	523
How to do it...	494	How to do it...	523
How it works...	499	How it works...	528
Configuring result set caching in Azure Synapse dedicated SQL pool	500	Optimizing query performance in Synapse Spark pools	528
Getting ready	500	Getting ready	529
How to do it...	500	How to do it...	529
How it works...	505	How it works...	534
Configuring longer backup retention for a Synapse SQL database	506		

13

Monitoring and Maintaining Azure Data Engineering Pipelines	535		
Technical requirements	536	How it works...	542
Monitoring Synapse integration pipelines using Log Analytics and workbooks	536	Tracing SQL queries for dedicated SQL pool to Synapse integration pipelines	543
Getting ready	536	Getting ready	543
How to do it...	537		

How to do it...	543	Getting ready	564
How it works...	549	How to do it...	565
Provisioning a Microsoft Purview account and creating a data catalog	549	How it works...	570
Getting ready	549	Applying Azure tags using PowerShell to multiple Azure resources	570
How to do it...	550	Getting ready	570
How it works...	564	How to do it...	570
Integrating a Synapse workspace with Microsoft Purview and tracking data lineage	564	How it works...	573
Index			575
Other Books You May Enjoy			584

Preface

Data is the new oil and probably the most valuable resource. Data engineering covers how one can gain insights out of data. This book will introduce the key processes in data engineering (ingesting, storing, processing, and consuming) and share a few common recipes that can help us develop data engineering pipelines to gain insights into our data.

The book follows the logical data engineering process by beginning with Azure Data Lake and covering data ingestion using Azure Data Factory into Azure Data Lake and Azure SQL Database in the first few chapters. In these chapters, the book also covers the management of common storage layers such as Azure Data Lake and Azure SQL Database, focusing on topics such as security, high availability, and performance monitoring. The middle chapters focus on data processing using Azure Databricks, Azure Synapse Analytics Spark pools, and Synapse dataflows, and data exploration using Synapse serverless SQL pools. The final few chapters focus on the consumption of the data using Synapse dedicated SQL pool and Synapse Spark lake databases, covering the tips and tricks to optimize and maintain Synapse dedicated SQL pool databases and lake databases. Finally, the book also has a bonus chapter on managing the overall data engineering pipeline, which covers pipeline monitoring using Azure Log Analytics and tracking data lineage using Microsoft Purview.

While the book can be consumed in parts or any sequence, following along sequentially will help the readers experience building an end-to-end data engineering solution on Azure.

Who this book is for

The book is for anyone working on data engineering projects in Azure. Azure data engineers, data architects, developers, and database administrators working on Azure will find the book extremely useful.

What this book covers

Chapter 1, Creating and Managing Data in Azure Data Lake, focuses on provisioning, uploading, and managing the data life cycle in Azure Data Lake accounts.

Chapter 2, Securing and Monitoring Data in Azure Data Lake, covers securing an Azure Data Lake account using firewall and private links, accessing data lake accounts using managed identities, and monitoring an Azure Data Lake account using Azure Monitor.

Chapter 3, Building Data Ingestion Pipelines Using Azure Data Factory, covers ingesting data using Azure Data Factory and copying data between Azure SQL Database and Azure Data Lake.

Chapter 4, Azure Data Factory Integration Runtime, focuses on configuring and managing self-hosted integration runtimes and running SSIS packages in Azure using Azure-SSIS integration runtimes.

Chapter 5, Configuring and Securing Azure SQL Database, covers configuring a Serverless SQL database, Hyperscale SQL database, and securing Azure SQL Database using virtual networks and private links.

Chapter 6, Implementing High Availability and Monitoring in Azure SQL Database, explains configuring high availability to Azure SQL Database using auto-failover groups and read replicas, monitoring Azure SQL Database, and the automated scaling of Azure SQL Database during utilization spikes.

Chapter 7, Processing Data Using Azure Databricks, covers integrating Azure Databricks with Azure Data Lake and Azure Key Vault, processing data using Databricks notebooks, working with Delta tables, and visualizing Delta tables using Power BI.

Chapter 8, Processing Data Using Azure Synapse Analytics covers exploring data using Synapse Serverless SQL pool, processing data using Synapse Spark Pools, Working with Synapse Lake database, and integrating Synapse Analytics with Power BI.

Chapter 9, Transforming Data Using Azure Synapse Dataflows, focuses on performing transformations using Synapse Dataflows, optimizing data flows using partitioning, and managing dynamic source schema changes using schema drifting.

Chapter 10, Building the Serving Layer in Azure Synapse SQL Pools, covers loading processed data into Synapse dedicated SQL pools, performing data archival using partitioning, managing table distributions, and optimizing performance using statistics and workload management.

Chapter 11, Monitoring Synapse SQL and Spark Pools, covers monitoring Synapse dedicated SQL and Spark pools using Azure Log Analytics workbooks, Kusto scripts, and Azure Monitor, and monitoring Synapse dedicated SQL pools using Dynamic Management Views (DMVs).

Chapter 12, Optimizing and Maintaining Synapse SQL and Spark Pools, offers techniques for tuning query performance by optimizing query plans, rebuilding replication caches and maintenance scripts to optimize Delta tables, and automatically pausing SQL pools during inactivity, among other things.

Chapter 13, Monitoring and Maintaining Azure Data Engineering Pipelines, covers monitoring and managing end-to-end data engineering pipelines, which includes tracking data lineage using Microsoft Purview and improving the observability of pipeline executions using log analytics and query labeling.

To get the most out of this book

Readers with exposure to Azure and a basic understanding of data engineering should easily be able to follow this book:

Software/hardware covered in the book	OS requirements
Azure subscription	Windows 10 or above
PowerShell 7 or above with Azure PowerShell installed	
SQL Server Management Studio installed	
Power BI Desktop installed	

If you are using the digital version of this book, we advise you to type the code yourself or access the code via the GitHub repository (link available in the next section). Doing so will help you avoid any potential errors related to the copying and pasting of code.

Download the example code files

You can download the example code files for this book from GitHub at <https://github.com/PacktPublishing/Azure-Data-Engineering-Cookbook-2nd-edition>. In case there's an update to the code, it will be updated in the existing GitHub repository.

We also have other code bundles from our rich catalog of books and videos available at <https://github.com/PacktPublishing/>. Check them out!

Download the color images

We also provide a PDF file that has color images of the screenshots and diagrams used in this book. You can download it here: <https://packt.link/CJshA>.

Conventions used

There are a number of text conventions used throughout this book.

Code in text: Indicates code words in text, database table names, folder names, filenames, file extensions, pathnames, dummy URLs, user input, and Twitter handles. Here is an example: "Observe that the `CopyFiles` package is now listed under the `AzureSSIS | Projects` folder."

A block of code is set as follows:

```
CREATE TABLE dbo.transaction_tbl WITH (DISTRIBUTION = ROUND_ROBIN)
AS
Select * from dbo.ext_transaction_tbl;
GO
Select TOP 100 * from dbo.transaction_tbl
GO
```

Any command-line input or output is written as follows:

Connect-AzAccount

Bold: Indicates a new term, an important word, or words that you see onscreen. For example, words in menus or dialog boxes appear in the text like this. Here is an example: "The **Configuration** section under the **Source** section of **Copy Data tool** can remain with defaults."

Tips or important notes

Appear like this.

Sections

In this book, you will find several headings that appear frequently (*Getting ready*, *How to do it...*, *How it works...*, *There's more...*, and *See also*).

To give clear instructions on how to complete a recipe, use these sections as follows.

Getting ready

This section tells you what to expect in the recipe and describes how to set up any software or any preliminary settings required for the recipe.

How to do it...

This section contains the steps required to follow the recipe.

How it works...

This section usually consists of a detailed explanation of what happened in the previous section.

There's more...

This section consists of additional information about the recipe in order to make you more knowledgeable about the recipe.

See also

This section provides helpful links to other useful information for the recipe.

Get in touch

Feedback from our readers is always welcome.

General feedback: If you have questions about any aspect of this book, mention the book title in the subject of your message and email us at customercare@packtpub.com.

Errata: Although we have taken every care to ensure the accuracy of our content, mistakes do happen. If you have found a mistake in this book, we would be grateful if you would report this to us. Please visit www.packtpub.com/support/errata, selecting your book, clicking on the Errata Submission Form link, and entering the details.

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 copyright@packt.com with a link to the material.

If you are interested in becoming an author: If there is a topic that you have expertise in and you are interested in either writing or contributing to a book, please visit authors.packtpub.com.

Share your thoughts

Once you've read *Azure Data Engineering Cookbook, Second Edition*, we'd love to hear your thoughts! Please click here to go straight to the Amazon review page for this book and share your feedback.

Your review is important to us and the tech community and will help us make sure we're delivering excellent quality content.

1

Creating and Managing Data in Azure Data Lake

Azure Data Lake is a highly scalable and durable object-based cloud storage solution from Microsoft. It is optimized to store large amounts of structured and semi-structured data such as logs, application data, and documents.

Azure Data Lake can be used as a data source and destination in data engineering projects. As a source, it can be used to stage structured or semi-structured data. As a destination, it can be used to store the result of a data pipeline.

Azure Data Lake is provisioned as a storage account in Azure, capable of storing files (blobs), tables, or queues. This book will focus on Azure Data Lake storage accounts used for storing blobs/files

In this chapter, we will learn how to provision, manage, and upload data into Data Lake accounts and will cover the following recipes:

- Provisioning an Azure storage account using the Azure portal
- Provisioning an Azure storage account using PowerShell

- Creating containers and uploading files to Azure Blob storage using PowerShell
- Managing blobs in Azure Storage using PowerShell
- Configuring blob lifecycle management for blob objects using the Azure portal

Technical requirements

For this chapter, the following are required:

- An Azure subscription
- Azure PowerShell

The code samples can be found at <https://github.com/PacktPublishing/Azure-Data-Engineering-Cookbook-2nd-edition>.

Provisioning an Azure storage account using the Azure portal

In this recipe, we will provision an Azure storage account using the Azure portal. Azure Blob storage is one of the four storage services available in Azure Storage. The other storage services are **Table**, **Queue**, and **File Share**. Table storage is used to store non-relational structured data as key-value pairs, queue storage is used to store messages as queues, and file share is used for creating file share directories/mount points that can be accessed using the NFS/SMB protocols. This chapter will focus on storing data using the Blob storage service.

Getting ready

Before you start, open a web browser and go to the Azure portal at <https://portal.azure.com>. Ensure that you have an Azure subscription. Install Azure PowerShell on your machine; instructions for installing it can be found at <https://docs.microsoft.com/en-us/powershell/azure/install-az-ps?view=azps-6.6.00>.

How to do it...

The steps for this recipe are as follows:

1. In the Azure portal, select **Create a resource** and choose **Storage account – blob, file, table, queue** (or search for **storage account** in the search bar; do not choose **Storage accounts (classic)**).
2. A new page, **Create a storage account**, will open. There are six tabs on the **Create a storage account** – **Basics**, **Advanced**, **Networking**, **Data protection**, **Tags**, and **Review + create**.
3. In the **Basics** tab, we need to provide the Azure **Subscription**, **Resource group**, **Storage account name**, **Region**, **Performance**, and **Redundancy** values, as shown in the following screenshot:

Create a storage account ...

Basics Advanced Networking Data protection Tags Review + create

Select the subscription in which to create the new storage account. Choose a new or existing resource group to organize and manage your storage account together with other resources.

Subscription *

Resource group * [Create new](#)

Instance details

If you need to create a legacy storage account type, please click [here](#).

Storage account name ⓘ *

Region ⓘ *

Performance ⓘ *
 Standard: Recommended for most scenarios (general-purpose v2 account)
 Premium: Recommended for scenarios that require low latency.

Redundancy ⓘ *
 Make read access to data available in the event of regional unavailability.

Review + create [< Previous](#) [Next : Advanced >](#)

Figure 1.1 – The Create a storage account Basics tab

4. In the **Advanced** tab, we need to select **Enable hierarchical namespace** under the **Data Lake Storage Gen2** settings:

The screenshot shows the 'Advanced' tab of the Azure Storage Account creation interface. It includes sections for 'Data protection', 'Tags', and 'Review + create'. Under 'Data protection', there are several checkboxes: 'Require secure transfer for REST API operations' (checked), 'Enable infrastructure encryption' (unchecked), 'Enable blob public access' (checked), 'Enable storage account key access' (checked), and 'Default to Azure Active Directory authorization in the Azure portal' (unchecked). A dropdown menu for 'Minimum TLS version' is set to 'Version 1.2'. Below this, the 'Data Lake Storage Gen2' section is expanded, showing a note about accelerating big data analytics workloads and enabling file-level access control lists (ACLS), followed by a checked checkbox for 'Enable hierarchical namespace'.

Figure 1.2 – Create a storage account – Advanced

5. In the **Networking** tab, we need to provide the connectivity method:

Create a storage account ...

The screenshot shows the 'Networking' tab of the Azure Storage Account creation interface. It includes tabs for 'Basics', 'Advanced', 'Networking' (which is selected and underlined), 'Data protection', 'Tags', and 'Review + create'. The 'Network connectivity' section contains a note about connecting via public or private endpoints. The 'Connectivity method' section shows three options: 'Public endpoint (all networks)' (selected with a blue dot), 'Public endpoint (selected networks)', and 'Private endpoint'. A detailed note explains that all networks will have access if 'Public endpoint (all networks)' is chosen, and it recommends using 'Private endpoint' for private access, with a link to 'Learn more'.

Figure 1.3 – Create a storage account – Networking

6. In the **Review + create** tab, review the configuration settings and select **Create** to provision the Azure storage account:

Home > Storage accounts >

Create a storage account ...

 Validation passed

Basics	Advanced	Networking	Data protection	Encryption	Tags	Review + create
Basics						
Subscription	Visual Studio Enterprise					
Resource Group	packtadestorage					
Location	eastus					
Storage account name	packtadestoragev2					
Deployment model	Resource manager					
Performance	Standard					
Replication	Read-access geo-redundant storage (RA-GRS)					
Advanced						
Secure transfer	Enabled					
Allow storage account key access	Enabled					
Allow cross-tenant replication	Disabled					
Default to Azure Active Directory authorization in the Azure portal	Disabled					
Blob public access	Enabled					
Minimum TLS version	Version 1.2					
Enable hierarchical namespace	Enabled					
Enable network file system v3	Disabled					
Access tier	Hot					
Enable SFTP	Disabled					
Large file shares	Disabled					

Figure 1.4 – Create a storage account – Review + create

How it works...

The Azure storage account is deployed in the selected subscription, resource group, and location. The **Performance** tier can be either **Standard** or **Premium**. A **Standard** performance tier is a low-cost magnetic drive-backed storage. It's suitable for applications such as static websites and bulk storing flat files. The **Premium** tier is a high-cost SSD-backed storage service. The **Premium** tier can only be used with Azure virtual machine disks for I/O-intensive applications.

The **Replication** options available are **Locally-redundant storage (LRS)**, **Zone-redundant storage (ZRS)**, **Geo-redundant storage (GRS)**, and **Geo-zone-redundant storage (GZRS)**. Local redundancy stores three local copies within the data center and provides fault tolerance for failures within it. Zone-redundant storage provides fault tolerance by copying data to additional data centers within the same region, while geo-redundant storage maintains copies across regions. Geo-zone-redundant storage combines geo- and zone-redundant features and offers the highest fault tolerance. The default **Geo-redundant storage (GRS)** option was selected, as it provides fault tolerance across regions.

Azure storage accounts can be accessed publicly over the internet, through selected networks (selected IPs and IP ranges), and from private endpoints.

Provisioning an Azure storage account using PowerShell

PowerShell is a scripting language used to programmatically manage various tasks. In this recipe, we will learn how to provision an Azure storage account using PowerShell.

Getting ready

Before you start, you need to log in to the Azure subscription from the PowerShell console. To do this, execute the following command in a new PowerShell window:

```
Connect-AzAccount
```

Then, follow the instructions to log in to the Azure account.

How to do it...

The steps for this recipe are as follows:

1. Execute the following command in a PowerShell window to create a new resource group. If you want to create the Azure storage account in an existing resource group, this step isn't required:

```
New-AzResourceGroup -Name Packtade-powershell -Location  
'East US'
```

You should get the following output:

```
ResourceGroupName : Packtade-powershell  
Location        : eastus  
ProvisioningState : Succeeded  
Tags            :  
ResourceId      : /subscriptions/b85b0984-a391-4f22-a832-fb6e46c39f38/resourceGroups/Packtade-powershell
```

Figure 1.5 – Creating a new resource group

2. Execute the following command to create a new Azure storage account in the Packtade-powershell resource group:

```
New-AzStorageAccount -ResourceGroupName Packtade-  
powershell -Name packtstoragepowershellv2 -SkuName  
Standard_LRS -Location 'East US' -Kind StorageV2
```

You should get the following output:

```
PS C:\Users\navenkat> New-AzStorageAccount -ResourceGroupName Packtade-powershell -Name packtstoragepowershellv2 -SkuName Standard_LRS -Loc  
ation 'East US' -Kind StorageV2  


| StorageAccountName       | ResourceGroupName   | PrimaryLocation | SkuName      | Kind      | AccessTier | CreationTime        | ProvisioningState | Enab<br>leHt<br>tpsT<br>raff<br>icOn<br>ly |
|--------------------------|---------------------|-----------------|--------------|-----------|------------|---------------------|-------------------|--------------------------------------------|
| packtstoragepowershellv2 | Packtade-powershell | eastus          | Standard_LRS | StorageV2 | Hot        | 6/2/2022 3:08:31 am | Succeeded         | True                                       |


```

Figure 1.6 – Creating a new storage account

How it works...

There is a single command to create an Azure storage account using PowerShell – `New-AzStorageAccount`. The `SkuName` parameter specifies the performance tier, and the `Kind` parameter specifies the account kind.

In the later recipes, we will look at how to assign public/private endpoints to an Azure storage account using PowerShell.

Creating containers and uploading files to Azure Blob storage using PowerShell

In this recipe, we will create a new container and upload files to Azure Blob storage using PowerShell.

Getting ready

Before you start, perform the following steps:

1. Make sure you have an existing Azure storage account. If not, create one by following the *Provisioning an Azure storage account using the Azure portal* recipe.
2. Log in to your Azure subscription in PowerShell. To log in, run the `Connect-AzAccount` command in a new PowerShell window and follow the instructions.

How to do it...

The steps for this recipe are as follows:

1. Execute the following commands to create the container in an Azure storage account:

```
$storageaccountname="packtadestoragev2"
$containername="logfiles"
$resourcegroup="packtadestorage"
#Get the Azure Storage account context
$storagecontext = (Get-AzStorageAccount -ResourceGroupName
$resourcegroup -Name $storageaccountname).Context;
#create a new container
New-AzStorageContainer -Name $containername -Context
$storagecontext
```

Container creation is usually very quick. You should get the following output:

```

PS C:\Users\navenkat> $storageaccountname="packtadestoragev2"
PS C:\Users\navenkat> $containername="logfiles"
PS C:\Users\navenkat> $resourcegroup="packtadestorage"
PS C:\Users\navenkat> Set the Azure Storage account context
PS C:\Users\navenkat> $storagecontext = (Get-AzStorageAccount -ResourceGroupName $resourcegroup -Name $storageaccountname).Context;
PS C:\Users\navenkat> #Create a new container
PS C:\Users\navenkat> New-AzStorageContainer -Name $containername -Context $storagecontext

Storage Account Name: packtadestoragev2

Name          PublicAccess      LastModified           IsDeleted  VersionId
----          -----          -----           -----      -----
logfiles      Off             6/2/2022 4:15:28 am +00:00

PS C:\Users\navenkat>

```

Figure 1.7 – Creating a new storage container

2. Execute the following commands to upload a text file to an existing container. Ensure that you create a folder in c:\\ADECookbook\\Chapter1\\Logfiles\\. Create any file inside the folder as Logfile1.txt:

```

#upload single file to container
Set-AzStorageBlobContent -File "C:\\ADECookbook\\Chapter1\\
Logfiles\\LogFile1.txt" -Context $storagecontext -Blob
LogFile1.txt -Container $containername

```

You should get an output similar to the following screenshot:

```

PS C:\Users\navenkat> #Upload single file to container
PS C:\Users\navenkat> Set-AzStorageBlobContent -File "C:\\ADECookbook\\Chapter1\\Logfiles\\LogFile1.txt" -Context $storagecontext -Blob logfile1.txt -Container $containername

AccountName: packtadestoragev2, ContainerName: logfiles

Name          BlobType    Length     ContentType          LastModified        AccessTier 'SnapshotTime
----          -----    15          application/octet-stream  2022-02-06 04:17:55Z Hot
logfile1.txt

```

Figure 1.8 – Uploading a file to a storage container

3. Execute the following commands to upload all the files in a directory to an Azure container. Create additional copies of Logfile1.txt in the same folder for testing multiple file uploads:

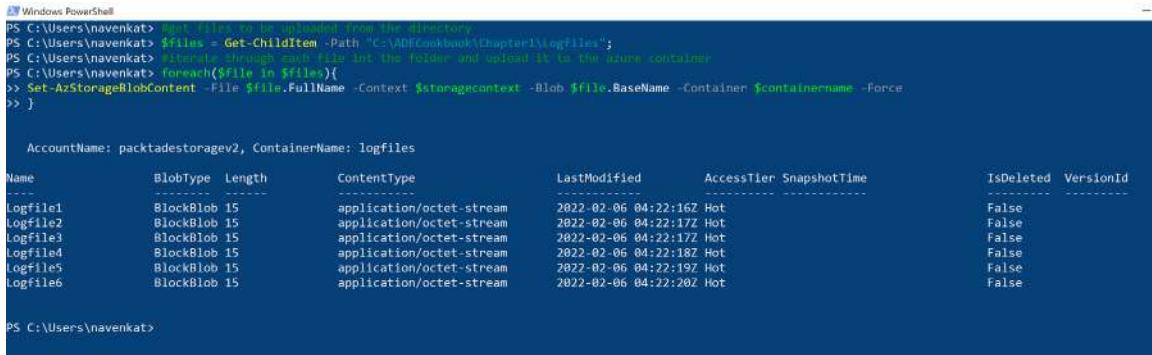
```

#get files to be uploaded from the directory
$files = Get-ChildItem -Path "C:\\ADECookbook\\Chapter1\\
Logfiles";
#iterate through each file int the folder and upload it
to the azure container
foreach($file in $files){

```

```
Set-AzStorageBlobContent -File $file.FullName -Context
$storagecontext -Blob $file.BaseName -Container
$containername -Force
}
```

You should get an output similar to the following screenshot:



```
PS C:\Users\navenkat> Get-ChildItem -Path "C:\A01Cookbook\Chapter1\logfiles";
PS C:\Users\navenkat> #upload all the files in the folder and upload it to the azure container
PS C:\Users\navenkat> foreach($file in $files){
>> Set-AzStorageBlobContent -File $file.FullName -Context $storagecontext -Blob $file.BaseName -Container $containername -Force
>> }

AccountName: packtadestoragev2, ContainerName: logfiles
Name          BlobType  Length    ContentType           LastModified      AccessTier SnapshotTime   IsDeleted  VersionId
----          -----  ----     -----           -----      -----           -----           -----       -----
logfile1      BlockBlob 15 application/octet-stream 2022-02-06 04:22:16Z Hot
logfile2      BlockBlob 15 application/octet-stream 2022-02-06 04:22:17Z Hot
logfile3      BlockBlob 15 application/octet-stream 2022-02-06 04:22:17Z Hot
logfile4      BlockBlob 15 application/octet-stream 2022-02-06 04:22:18Z Hot
logfile5      BlockBlob 15 application/octet-stream 2022-02-06 04:22:19Z Hot
logfile6      BlockBlob 15 application/octet-stream 2022-02-06 04:22:20Z Hot

PS C:\Users\navenkat>
```

Figure 1.9 – Uploading multiple files to a storage container

How it works...

The storage container is created using the `New-AzStorageContainer` command. It takes two parameters – the container name and the storage context. The storage context can be set using the `Get-AzStorageAccount` command context property.

To upload files to the container, we used the `Set-AzStorageBlobContent` command. This command requires the storage context, a file path to be uploaded, and the container name. To upload multiple files, we can iterate through the folder and upload each file using the `Set-AzStorageBlobContent` command.

Managing blobs in Azure Storage using PowerShell

In this recipe, we will learn how to perform various management tasks on an Azure blob. We will perform operations such as copying, listing, modifying, deleting, and downloading files from Azure Blob storage.

Getting ready

Before you start, perform the following steps:

1. Make sure you have an existing Azure storage account. If not, create one by following the *Provisioning an Azure storage account using PowerShell* recipe.

2. Make sure you have an existing Azure storage container. If not, create one by following the *Creating containers and uploading files to Azure Blob storage using PowerShell* recipe.
3. Log in to your Azure subscription in PowerShell. To log in, run the Connect-AzAccount command in a new PowerShell window and follow the instructions.

How to do it...

Let's perform the following operations in this recipe:

1. Copy files/blobs between two blob storage containers.
2. List files from a blob container.
3. Modify the storage access tier of a blob from **Hot** to **Cool**.
4. Download a file/blob from a container.
5. Delete a file/blob from a container.

Let's look at each of them in detail. We'll begin by copying blobs between containers.

Copying blobs between containers

Perform the following steps:

1. Execute the following commands to create a new container in an Azure storage account:

```
#set the parameter values
$storageaccountname="packtadestoragev2"
$resourcegroup="packtadestorage"
$sourcecontainername="logfiles"
$destcontainername="textfiles"
#Get storage account context
$storagecontext = (Get-AzStorageAccount -ResourceGroupName
$resourcegroup -Name $storageaccountname).Context
#create the container
$destcontainer = New-AzStorageContainer -Name
$destcontainername -Context $storagecontext
$destcontainer
```

You should get an output similar to the following screenshot:

```

PS C:\Users\navenkat> $storageaccountname="packtadestoragev2"
PS C:\Users\navenkat> $resourcegroup="packtadestorage"
PS C:\Users\navenkat> $sourcecontainername="Logfiles"
PS C:\Users\navenkat> $destcontainername="textfiles"
PS C:\Users\navenkat> #Get storage account context
PS C:\Users\navenkat> $storagecontext = (Get-AzStorageAccount -ResourceGroupName $resourcegroup -Name $storageaccountname).Context
PS C:\Users\navenkat> # create the container
PS C:\Users\navenkat> $destcontainer = New-AzStorageContainer -Name $destcontainername -Context $storagecontext
PS C:\Users\navenkat>
PS C:\Users\navenkat> $destcontainer

Storage Account Name: packtadestoragev2

Name          PublicAccess      LastModified           IsDeleted  VersionId
---          -----
textfiles     Off              6/2/2022 4:24:42 am +00:00

```

Figure 1.10 – Creating a new storage container

2. Execute the following command to copy the Logfile1 blob from the source container to the destination container:

```

#copy a single blob from one container to another

Start-CopyAzureStorageBlob -SrcBlob "LogFile1"
-SrcContainer $sourcecontainername -DestContainer
$destcontainername -Context $storagecontext -DestContext
$storagecontext

```

You should get an output similar to the following screenshot:

Name	BlobType	Length	ContentType	LastModified	AccessTier	SnapshotTime	IsDeleted	VersionId
LogFile1	BlockBlob	-1		2022-02-06 04:28:40Z			False	

Figure 1.11 – Copying a blob from one storage container to another

3. Execute the following command to copy all the blobs from the source container to the destination container:

```

# copy all blobs in new container

Get-AzStorageBlob -Container $sourcecontainername
-Context $storagecontext | Start-CopyAzureStorageBlob
-DestContainer $destcontainername -DestContext
$storagecontext -force

```

You should get an output similar to the following screenshot:

```
PS C:\Users\navenkat> Get-AzStorageBlob -Container $sourcecontainername -Context $storagecontext | Start-CopyAzureStorageBlob -DestContainer $destcontainername -DestContext $storagecontext -force

AccountName: packtadestoragev2, ContainerName: textfiles
Name          BlobType   Length    ContentType           LastModified      AccessTier SnapshotTime      IsDeleted  VersionId
----          -----   ----     -----           -----          -----          -----          -----      -----
logfile1      BlockBlob  15        application/octet-stream 2022-02-06 04:29:45Z Hot      False
logfile2      BlockBlob  15        application/octet-stream 2022-02-06 04:29:46Z       False
logfile3      BlockBlob  1      application/octet-stream 2022-02-06 04:29:46Z       False
logfile4      BlockBlob  -1       application/octet-stream 2022-02-06 04:29:46Z       False
logfile5      BlockBlob  -1       application/octet-stream 2022-02-06 04:29:46Z       False
logfile6      BlockBlob  -1       application/octet-stream 2022-02-06 04:29:46Z       False
logfile1.txt   BlockBlob  -1       application/octet-stream 2022-02-06 04:29:46Z       False

Ps C:\Users\navenkat>
```

Figure 1.12 – Copying all blobs from one storage container to another

Listing blobs in an Azure storage container

Execute the following command to list the blobs from the destination container:

```
# list the blobs in the destination container

(Get-AzStorageContainer -Name $destcontainername -Context
$storagecontext).CloudBlobContainer.ListBlobs()
```

You should get an output similar to the following screenshot:

```
PS C:\Users\navenkat> (Get-AzStorageContainer -Name $destcontainername -Context $storagecontext).CloudBlobContainer.ListBlobs()

Container Uri: https://packtadestoragev2.blob.core.windows.net/textfiles
Name          BlobType   Length    IsDeleted  RemainingDaysBeforePermanentDelete ContentType           LastModified      AccessTier SnapshotTime
----          -----   ----     -----      -----          -----           -----          -----          -----          -----
logfile1      BlockBlob  15        False      0             application/octet-stream 2022-02-06 04:29:45Z
logfile2      BlockBlob  15        False      0             application/octet-stream 2022-02-06 04:29:46Z
logfile3      BlockBlob  15        False      0             application/octet-stream 2022-02-06 04:29:46Z
logfile4      BlockBlob  15        False      0             application/octet-stream 2022-02-06 04:29:46Z
logfile5      BlockBlob  15        False      0             application/octet-stream 2022-02-06 04:29:46Z
logfile6      BlockBlob  15        False      0             application/octet-stream 2022-02-06 04:29:46Z
logfile1.txt   BlockBlob  -1       False      0             application/octet-stream 2022-02-06 04:29:46Z
```

Figure 1.13 – Listing blobs in a storage container

Modifying a blob access tier

Perform the following steps:

1. Execute the following commands to change the access tier of a blob:

```
# Get the blob reference
blob = Get-AzStorageBlob -Blob *LogFile2* -Container
$sourcecontainername -Context $storagecontext
#Get current access tier
blob
```

```
#change access tier to cool
$blob.ICloudBlob.SetStandardBlobTier("cool")
#Get the modified access tier
Get-AzStorageBlob -Blob *Logfile2* -Container
$sourcecontainername -Context $storagecontext
```

You should get an output similar to the following screenshot:

The screenshot shows a PowerShell session with the following commands and outputs:

```
PS C:\Users\navenkat> $blob = Get-AzStorageBlob -Blob "Logfile2" -Container $sourcecontainername -Context $storagecontext
PS C:\Users\navenkat> $blob
AccountName: packtadestoragev2, ContainerName: logfiles
Name          BlobType   Length    ContentType      LastModified      AccessTier SnapshotTime
---          -----   ----     -----      -----      -----      -----
Logfile2      BlockBlob  15       application/octet-stream 2022-02-06 04:22:17Z Hot
IsDeleted    VersionId
-----      -----
False         -----
```

PS C:\Users\navenkat> \$blob.ICloudBlob.SetStandardBlobTier("cool")
PS C:\Users\navenkat> \$blob
AccountName: packtadestoragev2, ContainerName: logfiles
Name BlobType Length ContentType LastModified AccessTier SnapshotTime
--- ----- ---- ----- ----- ----- -----
Logfile2 BlockBlob 15 application/octet-stream 2022-02-06 04:22:17Z Cool
IsDeleted VersionId
----- -----
False -----

Figure 1.14 – Modifying the blob access tier

2. Execute the following commands to change the access tier of all the blobs in the container:

```
#get blob reference
$blobs = Get-AzStorageBlob -Container $destcontainername
-Context $storagecontext
#change the access tier of all the blobs in the container
$blobs.ICloudBlob.setStandardBlobTier("Cool")
#verify the access tier
Get-AzStorageBlob -Container $destcontainername -Context
$storagecontext
```

You should get an output similar to the following screenshot:

```

PS C:\Users\navenkat> $destcontainername = "textfiles"
PS C:\Users\navenkat> $storagecontext = (Get-AzStorageAccount -ResourceGroupName "ADECookbook" -Name "packtadestoragev2").Context
PS C:\Users\navenkat> $blobs = Get-AzStorageBlob -Container $destcontainername -Context $storagecontext
PS C:\Users\navenkat> $blobs | Set-AzStorageBlob -AccessTier Cool
PS C:\Users\navenkat> Get-AzStorageBlob -Container $destcontainername -Context $storagecontext

```

AccountName: packtadestoragev2, ContainerName: textfiles								
Name	BlobType	Length	ContentType	LastModified	AccessTier	SnapshotTime	IsDeleted	VersionId
Logfile1	BlockBlob	15	application/octet-stream	2022-02-06 04:29:45Z	Cool		False	
Logfile2	BlockBlob	15	application/octet-stream	2022-02-06 04:29:46Z	Cool		False	
Logfile3	BlockBlob	15	application/octet-stream	2022-02-06 04:29:46Z	Cool		False	
Logfile4	BlockBlob	15	application/octet-stream	2022-02-06 04:29:46Z	Cool		False	
Logfile5	BlockBlob	15	application/octet-stream	2022-02-06 04:29:46Z	Cool		False	
Logfile6	BlockBlob	15	application/octet-stream	2022-02-06 04:29:46Z	Cool		False	
logfile1.txt	BlockBlob	15	application/octet-stream	2022-02-06 04:29:46Z	Cool		False	

Figure 1.15 – Modifying the blob access tier of all the blobs in a storage container

Downloading a blob

Execute the following commands to download a blob from Azure Storage to your local computer:

```

#get the storage context
$storagecontext = (Get-AzStorageAccount -ResourceGroupName "ADECookbook" -Name "packtadestoragev2").Context
$resourcegroup = $storagecontext.ResourceGroup
$storageaccountname = $storagecontext.StorageAccountName
$sourcecontainername = "textfiles"

#download the blob
Get-AzStorageBlobContent -Blob "Logfile1" -Container $sourcecontainername -Destination C:\ADECookbook\Chapter1\Logfiles\ -Context $storagecontext -Force

```

Deleting a blob

Execute the following command to remove/delete a blob:

```

#get the storage context
$storagecontext = (Get-AzStorageAccount -ResourceGroupName "ADECookbook" -Name "packtadestoragev2").Context
$resourcegroup = $storagecontext.ResourceGroup
$storageaccountname = $storagecontext.StorageAccountName
$sourcecontainername = "textfiles"

Remove-AzStorageBlob -Blob "Logfile2" -Container $sourcecontainername -Context $storagecontext

```

How it works...

Copying blobs across containers in the same storage account or a different storage account can be done easily by the PowerShell `Start-CopyAzureStorageBlob` command. The command takes the source and destination blobs, the source and destination containers, and the source and destination storage accounts as parameters. To copy all blobs in a container, we can run `Get-AzStorageBlob` to get all the blobs in the container and pipe the blobs to the `Start-CopyAzureStorageBlob` command.

A blob access tier can be modified by first getting the reference to the blob object using `Get-AzStorageBlob` and then modifying the access tier using the `SetStandardBlobTier` property. There are three access tiers – Hot, Cool, and Archive:

- The **Hot tier** is suitable for files that are accessed frequently. It has a higher storage cost and low access cost.
- The **Cool tier** is suitable for infrequently accessed files and has a lower access cost and a lower storage cost.
- The **Archive tier**, as the name suggests, is used for long-term archival and should be used for files that are seldom required. It has the highest access cost and the lowest storage cost.

To download a blob from Azure to a local system, we use `Get-AzStorageBlobContent`. The command accepts the blob name, the container name, the local file path, and the storage context.

To delete a blob, run `Remove-AzStorageBlob`. Provide the blob name, the container name, and the storage context.

Configuring blob lifecycle management for blob objects using the Azure portal

Azure Storage provides different blob access tiers such as **Hot, Cool, and Archive**. Each access tier has a different storage and data transfer cost. Applying a proper lifecycle rule to move a blob among different access tiers helps optimize the cost. In this recipe, we will learn how to apply a lifecycle rule to a blob using the Azure portal.

Getting ready

Before you start, perform the following steps:

1. Make sure you have an existing Azure storage account. If not, create one by following the *Provisioning an Azure storage account using PowerShell* recipe.
2. Make sure you have an existing Azure storage container. If not, create one by following the *Creating containers and uploading files to Azure Blob storage using PowerShell* recipe.
3. Make sure you have existing blobs/files in an Azure storage container. If not, you can upload blobs in accordance with the previous recipe. Then, log in to the Azure portal at <https://portal.azure.com>.

How to do it...

Follow the given steps to configure a blob lifecycle:

1. In the Azure portal, find and open the Azure Storage case. In our case, it is **packtadestoragev2**.
2. In the **packtadestoragev2** window, search for **Data management** and select **Lifecycle Management** under **Data management**, as shown in the following screenshot:



Figure 1.16 – Opening Lifecycle management

3. On the **Add a rule** page, create a rule to provide the lifecycle configuration. A lifecycle defines when to move a blob from a Hot to a Cool access tier, when to move a blob from a Cool to a Storage access tier, and when to delete the blob. Select **Limit blobs with filters** to create a lifecycle policy for a particular container. Click **Next**:

1 Details 2 Base blobs 3 Filter set

A rule is made up of one or more conditions and actions that apply to the entire storage account. Optionally, specify that rules will apply to particular blobs by limiting with filters.

Rule name *

Rule scope *

Apply rule to all blobs in your storage account

Limit blobs with filters

Blob type *

Block blobs

Append blobs

Blob subtype *

Base blobs

Snapshots

Versions

Figure 1.17 – Lifecycle management – the action set

4. Specify the condition you would like to use as a lifecycle policy. For example, the following rule moves the blobs that haven't been modified in the last 30 days to Cool storage and 60 days to Archive storage:

Add a rule ...

1 Details 2 Base blobs 3 Filter set

Lifecycle management uses your rules to automatically move blobs to cooler tiers or to delete them. If you create multiple rules, the associated actions must be implemented in tier order (from hot to cool storage, then archive, then deletion).

If

Base blobs were *

Last modified

More than (days ago) *

30

Then

Move to cool storage

If

Base blobs were *

Last modified

More than (days ago) *

60

Then

Move to archive storage

⚠ If you have workloads that require real-time read-access to these blobs, moving them to archive is not recommended. Blobs in archive must first be rehydrated to hot or cool to read them. [Learn more](#)

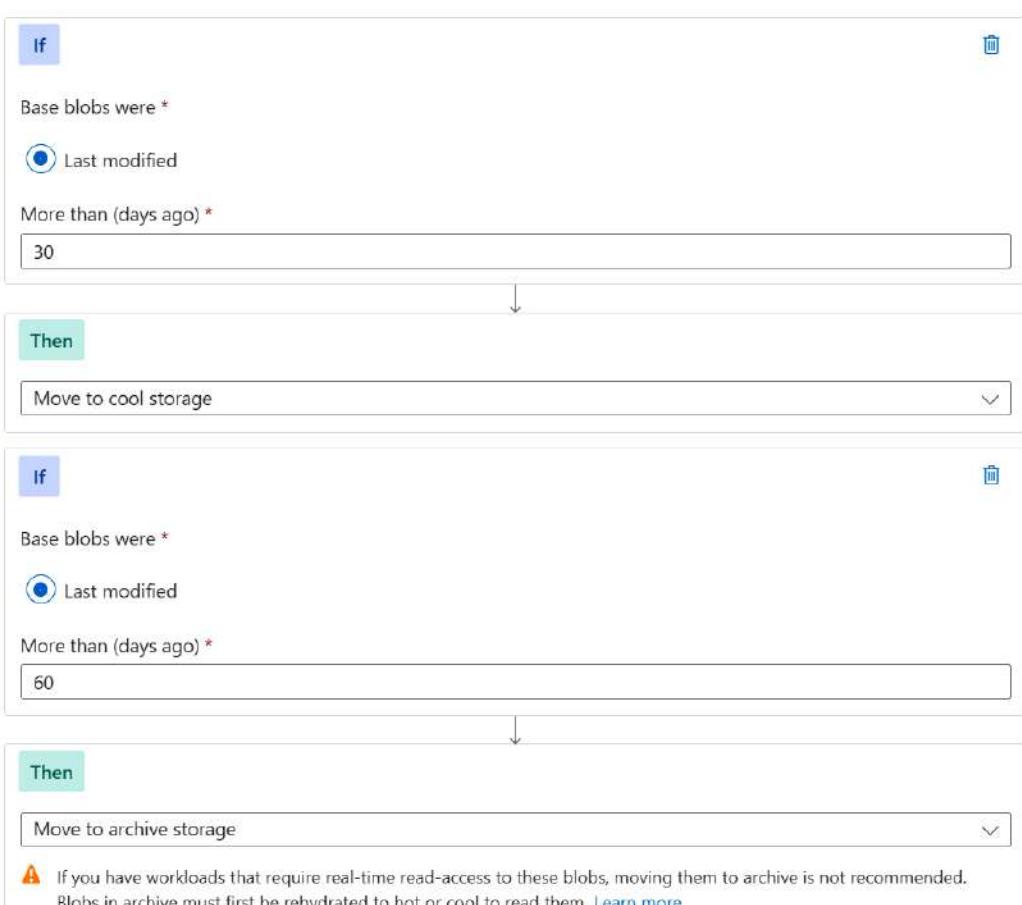


Figure 1.18 – Lifecycle management – the filter set

5. In **Filter set**, specify the specific container or filename for which you need to apply the rule:

Home > packtadestoragev2 >

Add a rule ...

Details Base blobs Filter set

Blob prefix

Filter blobs by name or first letters. To find items in a specific container, enter the name of the container followed by a forward slash, then the blob name or first letters. For example, to show all blobs starting with "a", type: "mycontainer/a".

Blob prefix

✖ 

Blob index match

If you have indexed items in containers with keys and values, you can filter for them.

Key	Value
<input type="text" value="Enter an index key"/>	<input type="text" value="=="/> <input type="button" value="▼"/> <input type="text" value="Enter a value"/>

Figure 1.19 – Lifecycle management – reviewing and adding

How it works...

A blob lifecycle management rule helps in managing storage costs by modifying the access tier of blobs as per the specified rule. Consider a log processing application that reads the log file from Azure Storage, analyzes it, and saves the result in a database. As the log file is read and processed, it may not be needed any further. Therefore, moving it to a Cool access tier from a Hot access tier will save on storage costs.

Blob lifecycle management helps in automating the access tier modification as per the application requirement and is, therefore, a must-have for any storage-based application.

2

Securing and Monitoring Data in Azure Data Lake

Data Lake forms the key storage layer for data engineering pipelines. Security and the monitoring of Data Lake accounts are key aspects of Data Lake maintenance. This chapter will focus on configuring security controls such as firewalls, encryption, and creating private links to a Data Lake account. By the end of this chapter, you will have learned how to configure a firewall, virtual network, and private link to secure the Data Lake, encrypt Data Lake using Azure Key Vault, and monitor key user actions in Data Lake.

We will be covering the following recipes in this chapter:

- Configuring a firewall for an Azure Data Lake account using the Azure portal
- Configuring virtual networks for an Azure Data Lake account using the Azure portal
- Configuring private links for an Azure Data Lake account
- Configuring encryption using Azure Key Vault for Azure Data Lake
- Accessing Blob storage accounts using managed identities
- Creating an alert to monitor an Azure Data Lake account
- Securing an Azure Data Lake account with an SAS using PowerShell

Configuring a firewall for an Azure Data Lake account using the Azure portal

Data Lake account access can be restricted to an IP or a range of IPs by whitelisting the allowed IPs in the storage account firewall. In this recipe, we'll learn to restrict access to a Data Lake account using a firewall.

Getting ready

Before you start, perform the following steps:

1. Open a web browser and go to the Azure portal at <https://portal.azure.com>.
2. Make sure you have an existing storage account. If not, create one using the *Provisioning an Azure storage account using the Azure portal* recipe in *Chapter 1, Creating and Managing Data in Azure Data Lake*.

How to do it...

To provide access to an IP or range of IPs, follow these steps:

1. In the Azure portal, locate and open the Azure storage account. In our case, the storage account is **packtadestoragev2**, created in the *Provisioning an Azure storage account using the Azure portal* recipe of *Chapter 1, Creating and Managing Data in Azure Data Lake*.
2. On the storage account page, in the **Security + Networking** section, locate and select **Firewalls and virtual networks**.

As the **packtadestoragev2** account was created with public access, it can be accessed from all networks.

3. To allow access from an IP or an IP range, click on the **Selected networks** option on the storage account on the **Firewalls and virtual networks** page:

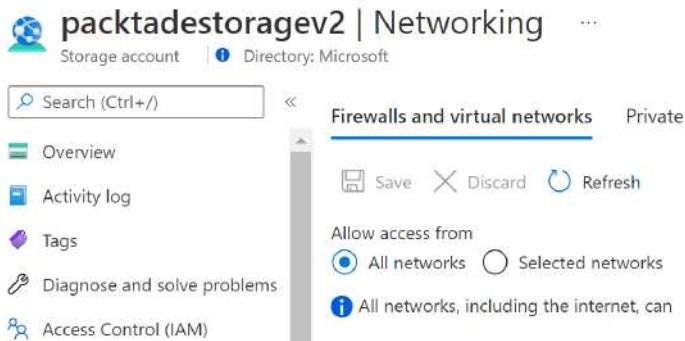


Figure 2.1 – Azure Storage – Firewalls and virtual networks

4. In the **Selected networks** option, scroll down to the **Firewall** section. To give access to your machine only, select the **Add your client IP** address option. To give access to a different IP or range of IPs, type in the IPs in the **Address range** section:

The screenshot shows the Azure Storage blade for the account 'packtadestoragev2'. The left sidebar has links for Overview, Activity log, Tags, Diagnose and solve problems, Access Control (IAM), Data migration, Events, Storage browser (preview), Data storage (Containers, File shares, Queues, Tables), and Security + networking (Networking, Access keys, Shared access signature, Encryption, Security). The main area is titled 'Firewalls and virtual networks' with a 'Private' tab selected. It shows a summary: 'Allow access from' with 'Selected networks' (radio button) selected. A tooltip says: 'Firewall settings allowing access to storage services will remain in effect for up to a minute after saving updated'. Under 'Virtual networks', there are buttons for 'Add existing virtual network' and 'Add new virtual network'. The 'Virtual Network' section shows 'No network selected.' In the 'Firewall' section, a red box highlights the 'Address range' input field containing 'IP address or CIDR' and the checked checkbox 'Add your client IP address ('116.89.64.165')'. Another red box highlights the 'Exceptions' section, which contains three checkboxes: 'Allow Azure services on the trusted services list to access this storage account.', 'Allow read access to storage logging from any network', and 'Allow read access to storage metrics from any network'. The last checkbox is unchecked.

Figure 2.2 – The whitelist IPs in the Azure Storage Firewall section

5. To access storage accounts from Azure services such as Azure Data Factory and Azure Functions, check **Allow Azure services on the trusted services list to access this storage account** under the **Exceptions** heading.
6. Click **Save** to save the configuration changes.

How it works...

Firewall settings are used to restrict access to an Azure storage account to an IP or range of IPs. Even if a storage account is public, it will only be accessible to the whitelisted IPs defined in the firewall configuration.

Configuring virtual networks for an Azure Data Lake account using the Azure portal

A storage account can be public which is accessible to everyone, public with access to an IP or range of IPs, or private with access to selected virtual networks. In this recipe, we'll learn how to restrict access to an Azure storage account in a virtual network.

Getting ready

Before you start, perform the following steps:

1. Open a web browser and go to the Azure portal at <https://portal.azure.com>.
2. Make sure you have an existing storage account. If not, create one using the *Provisioning an Azure storage account using the Azure portal* recipe in *Chapter 1, Creating and Managing Data in Azure Data Lake*.

How to do it...

To restrict access to a virtual network, follow the given steps:

1. In the Azure portal, locate and open the storage account. In our case, it's **packtadestoragev2**. On the storage account page, in the **Security + Network** section, locate and select **Firewalls and virtual networks | Selected networks**:

The screenshot shows the 'Networking' section of the Azure Storage blade for the 'packtadestoragev2' account. The left sidebar includes links for Overview, Activity log, Tags, Diagnose and solve problems, Access Control (IAM), Data migration, Events, and Storage browser (preview). The main area has tabs for 'Firewalls and virtual networks' (selected), 'Private endpoint connections', and 'Virtual networks'. Under 'Firewalls and virtual networks', there are buttons for Save, Discard, Refresh, and a radio button for 'Allow access from Selected networks' (which is highlighted with a red box). A note says 'Configure network security for your storage accounts.' Below this is a 'Virtual networks' section with 'Add existing virtual network' and 'Add new virtual network' buttons. The 'Virtual Network' and 'Subnet' tabs are shown, with 'No network selected.'

Figure 2.3 – Azure Storage – Selected networks

2. In the **Virtual networks** section, select + **Add new virtual network**:

This screenshot is identical to Figure 2.3, showing the 'Networking' section of the Azure Storage blade for the 'packtadestoragev2' account. The 'Add new virtual network' button in the 'Virtual networks' section is highlighted with a red box.

Figure 2.4 – Adding a virtual network

3. In the **Create virtual network** blade, provide the virtual network name, **Address space** details, and **Subnet** address range. The remainder of the configuration values are pre-filled, as shown in the following screenshot:

Create virtual network

Name *

 ✓

Address space * ⓘ

10.2.0.0/16 ✓
10.2.0.0 - 10.2.255.255 (65536 addresses)

Subscription *

Visual Studio Enterprise ✓

Resource group *

packtadestorage ✓
[Create new](#)

Location *

East US ✓

Subnet

Name *

Address range * ⓘ

10.2.0.0/24 ✓
10.2.0.0 - 10.2.0.255 (256 addresses)

DDoS protection ⓘ

Basic Standard

Service endpoint ⓘ

Microsoft.Storage

Firewall ⓘ

Disabled Enabled

Figure 2.5 – Creating a new virtual network

- Click on **Create** to create the virtual network. This is created and listed in the **Virtual Network** section, as shown in the following screenshot:

The screenshot shows the 'Firewalls and virtual networks' tab selected in the Azure portal. At the top, there are 'Save', 'Discard', and 'Refresh' buttons. Below that, it says 'Allow access from' with 'Selected networks' (radio button) selected. There is a note to 'Configure network security for your storage accounts' with a 'Learn more' link. Under 'Virtual networks', there are 'Add existing virtual network' and 'Add new virtual network' buttons. A table lists a single entry:

Virtual Network	Subnet	Address range	Endpoint Status
packtvenet	1		

Figure 2.6 – Saving a virtual network configuration

- Click **Save** to save the configuration changes.

How it works...

We first created an Azure virtual network and then added it to the Azure storage account. Creating the Azure virtual network from the storage account page automatically fills in the resource group, location, and subscription information. The virtual network and the storage account should be in the same location.

The address space specifies the number of IP addresses in a given virtual network.

We also need to define the subnet within the virtual network that the storage account will belong to. We can also create a custom subnet. In our case, for the sake of simplicity, we have used the default subnet.

This allows the storage account to only be accessed by resources that belong to the given virtual network. The storage account is inaccessible to any network other than the specified virtual network.

Configuring private links for an Azure Data Lake account

In this recipe, we will be creating a private link to a storage account and using private endpoints to connect to it.

Private links and private endpoints ensure that all communication to the storage account goes through the Azure backbone network. Communications to the storage account don't use a public internet network, which makes them very secure.

Getting ready

Before you start, perform the following steps:

1. Open a web browser and go to the Azure portal at <https://portal.azure.com>.
2. Make sure you have an existing storage account. If not, create one using the *Provisioning an Azure storage account using the Azure Portal* recipe in *Chapter 1, Creating and Managing Data in Azure Data Lake*.
3. Make sure you have an existing virtual network configured to the storage account. If not, create one using the *Configuring virtual networks for an Azure Data Lake account using the Azure portal* recipe in this chapter.

How to do it...

Perform the following steps to configure private links to a Data Lake account:

1. Log in to the Azure portal and click on the storage account.
2. Click on **Networking** | the **Private Endpoints** tab.
3. Click on the **+ Private endpoint** button, as shown here:

Home > packtadestoragev2_1639820964889 > packtadestorage > packtadestoragev2

The screenshot shows the 'Networking' blade for the 'packtadestoragev2' storage account. On the left, there's a sidebar with links: Overview, Activity log, Tags, Diagnose and solve problems, and Access Control (IAM). The main area has tabs: Firewalls and virtual networks and Private endpoint connections (which is highlighted with a red box). Below these are buttons for '+ Private endpoint' (also highlighted with a red box), Approve, Reject, Remove, Refresh, and a Filter by name... input field. A dropdown menu shows 'All connection states'. At the bottom, there's a 'Connection state' section and a message 'No results'.

Figure 2.7 – Creating a private endpoint to a storage account

4. Provide an endpoint name, as shown in the following screenshot:

Home > packtadestoragev2_1639820964889 > packtadestorage > packtadestoragev2 >

The screenshot shows the 'Create a private endpoint' wizard at the 'Basics' step. It has five tabs: Basics (selected), Resource, Configuration, Tags, and Review + create. The 'Project details' section includes a 'Subscription' dropdown set to 'Visual Studio Enterprise' and a 'Resource group' dropdown set to 'packtadestorage'. The 'Instance details' section includes a 'Name' field containing 'packtadeprivateendpoint' and a 'Region' dropdown set to 'East US'.

Figure 2.8 – Providing an endpoint name

5. In the **Resource** tab, set **Target sub-resource** to **dfs**. **Distributed File Systems (DFS)** is sub-source if we are connecting to Data Lake Storage Gen2. The rest of the fields are auto-populated. Proceed to the **Configuration** section:

Home > packtadestoragev2_1639820964889 > packtadestorage > packtadestoragev2 >

Create a private endpoint ...

Basics **2 Resource** Configuration Tags Review + create

Private Link offers options to create private endpoints for different Azure resources, like your private link service, a SQL server, or an Azure storage account. Select which resource you would like to connect to using this private endpoint. [Learn more](#)

Subscription

Visual Studio Enterprise

Resource type

Microsoft.Storage/storageAccounts

Resource

packtadestoragev2

Target sub-resource * ⓘ

dfs



Figure 2.9 – Setting the target resource type to dfs

6. Create a private **Domain Name System (DNS)** zone by picking the same resource group where you created the storage account, as shown in the following screenshot:

Home > packtadestoragev2 > packtadestorage > packtadestoragev2 >

Create a private endpoint ...

Basics Resource **3 Configuration** Tags Review + create

Networking

To deploy the private endpoint, select a virtual network subnet. [Learn more](#)

Virtual network * ⓘ

packvnet



Subnet * ⓘ

packvnet/default (10.2.0.0/24)



! If you have a network security group (NSG) enabled for the subnet above, it will be disabled for private endpoints on this subnet only. Other resources on the subnet will still have NSG enforcement.

Private DNS integration

To connect privately with your private endpoint, you need a DNS record. We recommend that you integrate your private endpoint with a private DNS zone. You can also utilize your own DNS servers or create DNS records using the host files on your virtual machines. [Learn more](#)

Integrate with private DNS zone

Yes No

Configuration name

Subscription

Resource group

Private DNS zone

privatelink-dfs-core-windows-net

Visual Studio Enterprise

packtadestorage

(new) privatelink.dfs.core.windows.net

Figure 2.10 – Creating a private DNS

7. Hit the **Create** button to create the private DNS link.
8. After the private endpoint is created, open it in the Azure portal. Click on **DNS configuration**:

The screenshot shows the Azure portal interface for managing a private endpoint named 'packtadepivateendpoint'. The left sidebar has a 'DNS' icon selected. The main content area is titled 'DNS configuration'. It includes sections for 'Private DNS integration' (with a note about creating a DNS record) and 'Customer Visible FQDNs' (listing 'packtadepivateendpoint.nic.e7fc084e-5c23...'). A table lists network interfaces with their IP addresses and FQDNs. The IP address '10.2.0.4' and the FQDN 'packtadestoragev2.dfs.core.windows.net' are highlighted with red boxes. The 'DNS configuration' item in the sidebar is also highlighted with a red box.

Figure 2.11 – Copy the FQDN

- Make a note of the **FQDN** and **IP addresses** details. The **FQDN** is the **Fully Qualified Domain Name**, which will resolve to the private IP address if, and only if, you are connected to the virtual network.

With the preceding steps, we have created a private endpoint that will use private links to connect to a storage account.

How it works...

We have created a private link to a storage account and ensured that traffic goes through the Microsoft backbone network (and not the public internet), as we will be accessing the storage account via a private endpoint. To show how it works, let's resolve the private URL link from the following locations. Let's perform the following:

- Use nslookup to look up a private URL link from your local machine.
- Use nslookup to look up a private URL link from a virtual machine inside the virtual network.

On your machine, open Command Prompt and type nslookup <FQDN of private link>, as shown in the following screenshot:

```
C:\Users\navenkat>nslookup packtadestoragev2.dfs.core.windows.net
Server: UnKnown
Address: 2404:e801:2000:50d:3223:3ff:fe4:c8e2

Non-authoritative answer:
Name:   dfs.mnz22prdstr03a.store.core.windows.net
Address: 20.60.128.226
Aliases: packtadestoragev2.dfs.core.windows.net
          packtadestoragev2.privatelink.dfs.core.windows.net
```

Figure 2.12 – Testing a private endpoint connection outside of the virtual network

nslookup resolves the private link to an incorrect IP address, as your machine is not part of the virtual network. To see it working, perform the following instructions:

1. Create a new virtual machine in the Azure portal. Ensure to allow a remote desktop connection to the virtual machine, as shown in the following screenshot:

Inbound port rules

Select which virtual machine network ports are accessible from the public internet. You can specify more limited or granular network access on the Networking tab.

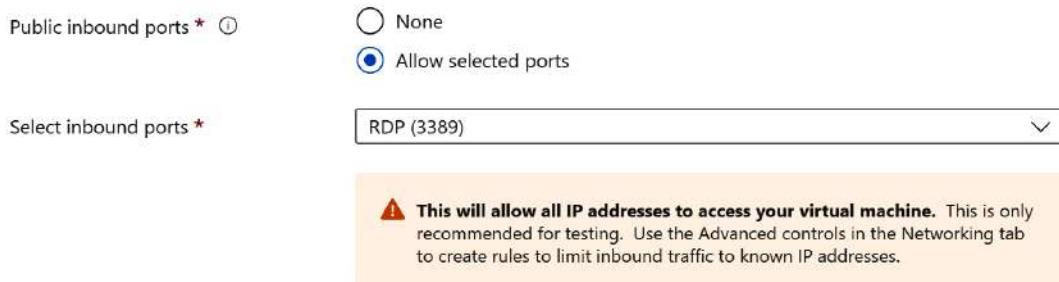


Figure 2.13 – Creating a new virtual machine and allowing a remote desktop

2. Under **Networking**, select the virtual network in which the storage account resides:

Basics Disks **Networking** Management Advanced Tags Review + create

Define network connectivity for your virtual machine by configuring network interface card (NIC) settings. You can control ports, inbound and outbound connectivity with security group rules, or place behind an existing load balancing solution.
[Learn more](#)

Network interface

When creating a virtual machine, a network interface will be created for you.

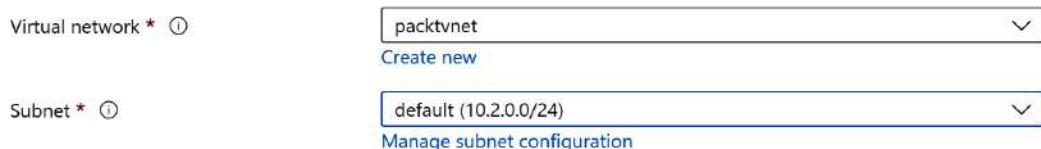


Figure 2.14 – Configuring the virtual machine to use the virtual network

Once the virtual machine is created, log in to the virtual machine using a remote desktop and perform nslookup to look up the private link URL again to resolve its IP address. nslookup is a command that will resolve an URL to an IP address. We will use nslookup to verify whether the private link URL resolves to a private IP address (10.x.x.x) and not a public IP address.

nslookup from a virtual machine inside the virtual network resolves correctly to the private IP address of the private link, as shown in the following screenshot. This shows that the connection goes through a virtual network only and doesn't use public internet:

```
C:\Users\rajacct>nslookup packtadestoragev2.dfs.core.windows.net
Server: UnKnown
Address: 168.63.129.16

Non-authoritative answer:
Name: packtadestoragev2.privatelink.dfs.core.windows.net
Address: 10.2.0.4
Aliases: packtadestoragev2.dfs.core.windows.net
```

Figure 2.15 – nslookup from the virtual network

With the previous recipe, we have successfully created a private link to a storage account, configured a private endpoint connection, and accessed it via a virtual machine to verify the connectivity. This recipe covers how you can securely connect to a storage account through virtual networks only by passing a public network.

Configuring encryption using Azure Key Vault for Azure Data Lake

In this recipe, we will create a key vault and use it to encrypt an Azure Data Lake account.

Azure Data Lake accounts are encrypted at rest by default using Azure managed keys. However, you have the option of bringing your own key to encrypt an Azure Data Lake account. Using your own key gives better control over encryption.

Getting ready

Before you start, perform the following steps:

1. Open a web browser and go to the Azure portal at `https://portal.azure.com`.
2. Make sure that you have an existing storage account. If not, create one using the *Provisioning an Azure storage account using the Azure portal* recipe in *Chapter 1, Creating and Managing Data in Azure Data Lake*.

How to do it...

Perform the following steps to add encryption to a Data Lake account using Azure Key Vault:

1. Log in to portal.azure.com, click on **Create a resource**, search for **Key Vault**, and click on **Create**. Provide the key vault details, as shown in the following screenshot. Click on **Review + Create**:

Home > Create a resource > Key Vault >

Create a key vault ...

Basics Access policy Networking Tags Review + create

Azure Key Vault is a cloud service used to manage keys, secrets, and certificates. Key Vault eliminates the need for developers to store security information in their code. It allows you to centralize the storage of your application secrets which greatly reduces the chances that secrets may be leaked. Key Vault also allows you to securely store secrets and keys backed by Hardware Security Modules or HSMs. The HSMs used are Federal Information Processing Standards (FIPS) 140-2 Level 2 validated. In addition, key vault provides logs of all access and usage attempts of your secrets so you have a complete audit trail for compliance.

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

Resource group * packtadestorage

[Create new](#)

Instance details

Key vault name * PackAdeKeyVault

Region * East US

Pricing tier * Standard

Recovery options

Soft delete protection will automatically be enabled on this key vault. This feature allows you to recover or permanently delete a key vault and secrets for the duration of the retention period. This protection applies to the key vault and the secrets stored within the key vault.

Figure 2.16 – Creating an Azure key vault

2. Go to the storage account to be encrypted. Search for Encryption on the left. Click on **Encryption** and select **Customer-managed keys** as the **Encryption type**. Click on **Select a key vault and key** at the bottom:

Home > PacktAdeKeyVault > PacktAdeKeyVault > packtadestorage > packtadestoragev2

The screenshot shows the 'Encryption' blade for the 'packtadestoragev2' storage account. The 'Encryption' tab is selected. In the 'Encryption selection' section, 'Customer-managed keys' is chosen. In the 'Key selection' section, 'Select from key vault' is chosen. A callout box highlights the 'Select a key vault and key' button.

packtadestoragev2 | Encryption

Storage account | Directory: Microsoft

enr X « Encryption Encryption scopes ...

Security + networking

Access keys

Encryption

Encryption selection

Enable support for customer-managed keys Blobs and files only

Infrastructure encryption Disabled

Encryption type Customer-managed keys Microsoft-managed keys

When customer-managed keys are enabled, purge protection are also enabled on the key

Key selection

Encryption key Select from key vault Enter key URI

Key vault and key *

Figure 2.17 – Encrypting using customer-managed keys

3. On the new screen, **Select a key**, select **Key store type** as **Key vault** and select the newly created **PacktAdeKeyVault** as **Key vault**. Click on **Create new key**, as shown in the following screenshot:

Home > PacktAdeKeyVault > PacktAdeKeyVault > packtadestorage > packtadestoragev2 >

Select a key ...

Subscription *

Key store type Key vault Managed HSM

Key vault * [Create new key vault](#)

Key * [Create new key](#)

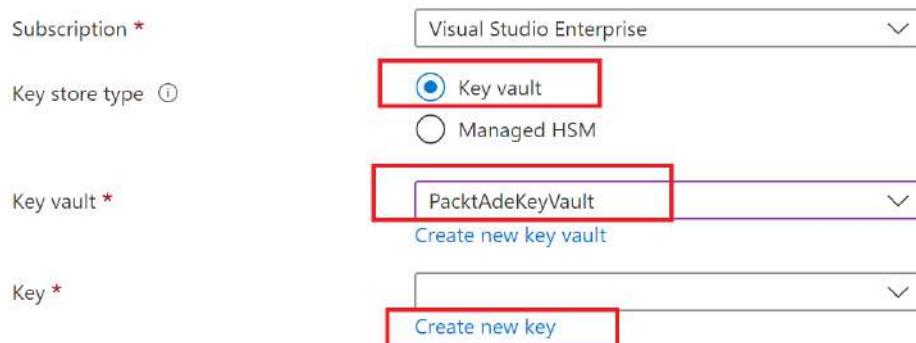


Figure 2.18 – Selecting Key Vault

4. Provide a name for the key to be used for encryption of the storage account. The default option, **Generate**, ensures that the key is generated automatically. Click on **Create**:

Home > PacktAdeKeyVault > PacktAdeKeyVault > packtadestorage > packtadestoragev2 > Select a key >

Create a key ...

Options

Name * 

Key type RSA EC

RSA key size 2048 3072 4096

Set activation date

Set expiration date

Enabled Yes No

Tags

Set key rotation policy (Preview)

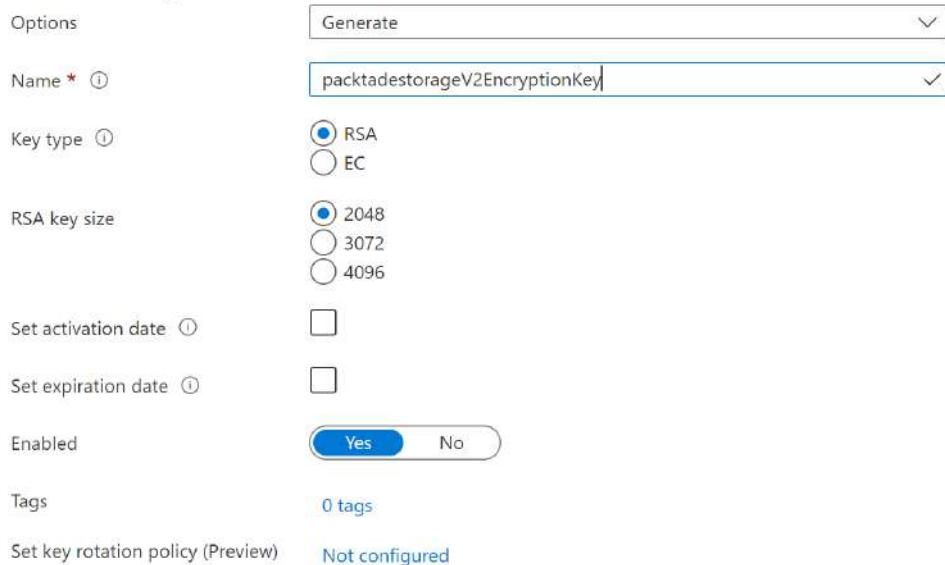


Figure 2.19 – Creating a key

5. Once the key is created, the screen automatically moves to the key vault selection page in the Blob storage, and the newly created key is selected as the key. Click on **Select**:

Home > PacktAdeKeyVault > PacktAdeKeyVault > packtadestorage > packtadestoragev2 >

Select a key

The screenshot shows the 'Select a key' page in the Azure portal. At the top, there is a success message: 'The key 'packtadestorageV2EncryptionKey' has been successfully created.' Below this, there are three input fields:

- Subscription ***: A dropdown menu showing 'Visual Studio Enterprise'.
- Key store type**: A radio button group where 'Key vault' is selected (indicated by a blue dot) and 'Managed HSM' is unselected.
- Key vault ***: A dropdown menu showing 'PacktAdeKeyVault' with a 'Create new key vault' link below it.

Below these fields is another dropdown menu labeled 'Key *' which contains the value 'packtadestorageV2EncryptionKey'. This dropdown is highlighted with a red rectangular box. There is also a 'Create new key' link below it.

Figure 2.20 – Selecting the key

6. The screen moves to the encryption page on the Blob storage page. Click on **Save** to complete the encryption configuration.

How it works...

As the newly created key vault has been set for encryption on an Azure Data Lake account, all Data Lake operations (`read`, `write`, and `metadata`) will use the key from Key Vault to encrypt and decrypt the data in Data Lake. The encryption and decryption operations are fully transparent and have no impact on users' operations.

The Data Lake account automatically gets permissions on the key vault to extract the key and perform encryption on data. You can verify this by *opening the key vault* in the Azure portal and clicking on **Access Policies**. Note that the storage account has been granted **Get**, **wrap**, and **unwrap** permissions on the keys, as shown in the next screenshot:

The screenshot shows the 'Access policies' section of the Azure Key Vault interface. On the left, a sidebar lists navigation options like Overview, Activity log, Access control (IAM), Tags, Diagnose and solve problems, Events, and Settings (Keys, Secrets, Certificates, Access policies, Networking, Security). The 'Access policies' option is currently selected. The main area displays a table of access policies:

Name	Email
APPLICATION	packtadestoragev2

Below the table, a large modal window titled 'Access policies' is open. It contains sections for 'Enable Access to:' (checkboxes for Azure VM deployment, ARM template deployment, and Disk Encryption), 'Permission model' (radio buttons for 'Vault access' and 'Azure role-based'), and 'Cryptographic Operations' (checkboxes for Decrypt, Encrypt, Unwrap Key, and Wrap Key). The 'Unwrap Key' and 'Wrap Key' checkboxes are checked. A sidebar on the right shows a list of permissions with checkboxes: Select all, Get (checked), List, Update, Create, Import, Delete, Recover, Backup, and Restore. The 'Get' checkbox is checked. At the bottom of the modal, a dropdown menu shows '3 selected'.

Figure 2.21 – Storage account permissions in Key Vault

Accessing Blob storage accounts using managed identities

In this recipe, we will grant permissions to managed identities on a storage account and showcase how you can use managed identities to connect to Azure Data Lake.

Managed identities are password-less service accounts used by Azure services such as Data Factory and Azure VMs to access other Azure services, such as Blob storage. In this recipe, we will show you how Azure Data Factory's managed identity can be granted permission on an Azure Blob storage account.

Getting ready

Before you start, perform the following steps:

1. Open a web browser and go to the Azure portal at <https://portal.azure.com>.
2. Make sure you have an existing storage account. If not, create one using the *Provisioning an Azure storage account using the Azure portal* recipe in *Chapter 1, Creating and Managing Data in Azure Data Lake*.

How to do it...

We will be testing accessing a Data Lake account using managed identities. To achieve this, we will create a Data Factory account and use Data Factory's managed identity to access the Data Lake account. Perform the following steps to test this:

1. Create an Azure Data Factory by using the following PowerShell command:

```
$resourceGroupName = " packtadestorage";
$location = 'east us'
$dataFactoryName = "ADFPacktADE2";
$DataFactory = Set-AzDataFactoryV2 -ResourceGroupName
$resourceGroupName -Location $location -Name
$dataFactoryName
```

2. Go to the storage account in the Azure portal. Click on **Access Control (IAM)** and then **Add**, as shown in the following screenshot:

The screenshot shows the 'Access Control (IAM)' blade for a Storage account named 'packtadestoragev2'. On the left, there's a sidebar with links like Overview, Activity log, Tags, Diagnose and solve problems, Access Control (IAM), Data migration, and Events. The 'Access Control (IAM)' link is highlighted with a red box. At the top right, there are buttons for 'Add' (also highlighted with a red box), 'Download role assignments', and 'Edit colu...'. Below these, tabs for 'Check access', 'Role assignments', 'Roles', and 'Deny' are present, with 'Check access' being the active tab. Under 'My access', it says 'View my level of access to this resource.' and has a 'View my access' button. The 'Check access' section below it says 'Review the level of access a user, group, service principal, or managed identity has to this resource.' with a 'Learn more' link.

Figure 2.22 – Adding a role to a managed identity

3. Select **Add role assignment** and search for the Storage Blob Data Contributor role. Select the role and click **Next**. Select **Managed identity** in **Assign access to** and click on **+ Select members**, as shown in the following screenshot:

The screenshot shows the 'Add role assignment' blade. At the top, there's a feedback link and tabs for 'Role', 'Members' (which is selected and highlighted with a red box), 'Conditions (optional)', and 'Review + assign'. Below this, the 'Selected role' is set to 'Storage Blob Data Contributor'. The 'Assign access to' section shows two options: 'User, group, or service principal' (radio button) and 'Managed identity' (radio button, which is selected and highlighted with a red box). At the bottom, there's a 'Members' section with a '+ Select members' button (highlighted with a red box).

Figure 2.23 – Selecting the Data Factory managed identity

4. Your subscription should be selected by default. From the **Managed identity** dropdown, select **Data Factory (V2) (1)**. Select the recently created **ADFPacktADE2** Data Factory and click on the **Select** button:

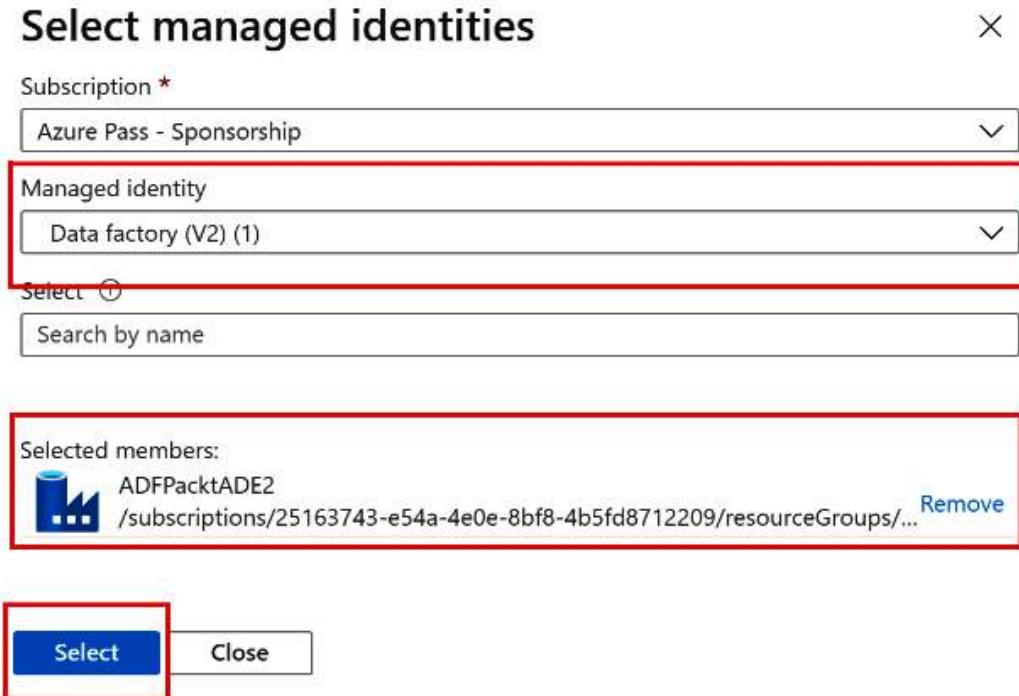


Figure 2.24 – Assigning a role to a managed identity

5. Click on **Review + Assign** to complete the assignment. To test whether it's working, open the **ADFPacktADE2** Data Factory that was created in *step 1*. Click on **Open Azure Data Factory Studio**, as shown in the next screenshot:

The screenshot shows the Azure Data Factory (V2) interface. On the left, there's a navigation sidebar with the following items:

- Overview (selected)
- Activity log
- Access control (IAM)
- Tags
- Diagnose and solve problems

Below this is a 'Settings' section:

- Networking
- Managed identities (selected)
- Properties
- Locks

Under 'Getting started':

- Quick start

The main content area has a heading 'Essentials' with the following details:

- Resource group (change) : PacktADE
- Status : Succeeded
- Location : East US
- Subscription (change) : Azure Pass - Sponsorship
- Subscription ID :

Below this is a 'Getting started' section with a button:

Open Azure Data Factory Studio
Start authoring and monitoring your data pipelines and data flows.
[Open](#)

A red box highlights the 'Open Azure Data Factory Studio' button.

Figure 2.25 – Opening Azure Data Factory Studio

6. Click on the **Manage** button on the left and then **Linked services**. Click on **+ New**, as shown in the following screenshot:

The screenshot shows the 'Data Factory' blade in the Azure portal. On the left, there's a sidebar with the following items:

- Home
- Connections (selected)
- Integration runtimes
- Azure Purview
- Source control

The main content area has a heading 'Linked services' with the following sub-section:

Linked service defines the

[+ New](#)

Filter by name

Showing 0 - 0 of 0 items

Red boxes highlight the 'Connections' item in the sidebar, the 'Linked services' item in the main content, and the '+ New' button.

Figure 2.26 – Creating a linked service in Data Factory

7. Search for Data Lake and select **Azure Data Lake Storage Gen 2** as the data store. Select **Managed Identity** for **Authentication method**. Select the storage account (**packadestoragev2**) for **Storage account name**. Click on **Test connection**:

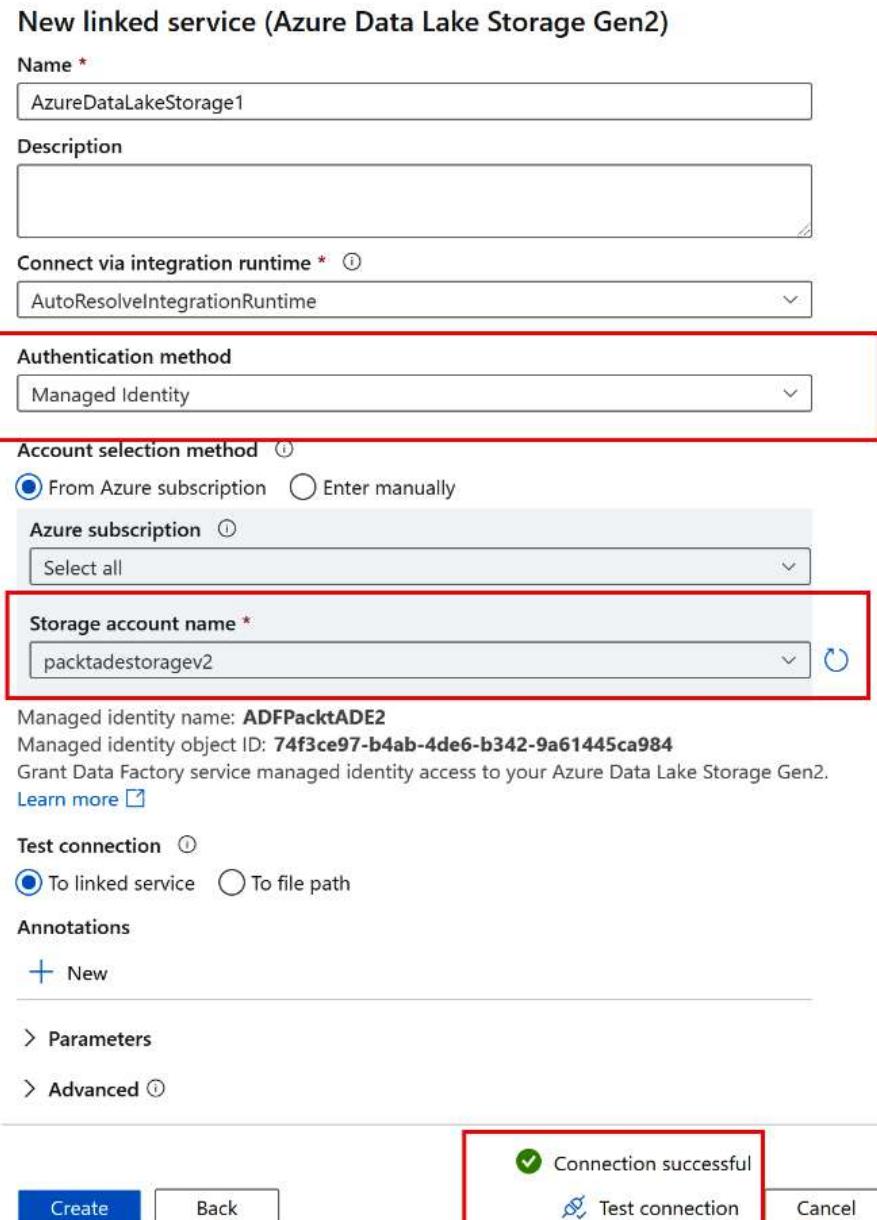


Figure 2.27 – Testing a managed identity connection in Data Factory

A successful test connection indicates that we can successfully connect to a storage account using a managed identity.

How it works...

A managed identity for the data factory was automatically created when the Data Factory account was created. We provided the **Storage Blob Data Contributor** permission on the Azure Data Lake storage account to the managed identity of Data Factory. Hence, Data Factory was successfully able to connect to the storage account in a secure way without using a key/password.

Creating an alert to monitor an Azure storage account

We can create an alert on multiple available metrics to monitor an Azure storage account. To create an alert, we need to define the trigger condition and the action to be performed when the alert is triggered. In this recipe, we'll create an alert to send an email if the used capacity metrics for an Azure storage account exceed 5 MB. The used capacity threshold of 5 MB is not a standard and is deliberately kept low to explain the alert functionality.

Getting ready

Before you start, perform the following steps:

1. Open a web browser and log in to the Azure portal at <https://portal.azure.com>.
2. Make sure you have an existing storage account. If not, create one using the *Provisioning an Azure storage account using the Azure portal* recipe in *Chapter 1, Creating and Managing Data in Azure Data Lake*.

How to do it...

Follow these steps to create an alert:

1. In the Azure portal, locate and open the storage account. In our case, the storage account is **packtadestoragev2**. On the storage account page, search for **alert** and open **Alerts** in the **Monitoring** section:

The screenshot shows the Azure portal interface. At the top, there is a breadcrumb navigation bar: Home > packtadestoragev2. Below this, the storage account name 'packtadestoragev2' is displayed with a green exclamation mark icon and the text 'Storage account'. A search bar contains the text 'alert'. Under the 'Monitoring' heading, a 'Alerts' button is highlighted with a green background and white text. The URL in the browser's address bar is 'https://portal.azure.com/#blade/HubsBlade/search/alert'.

Figure 2.28 – Selecting Alerts

2. On the **Alerts** page, click on **+ New alert rule**:

The screenshot shows the 'Alerts' page for the 'packtadestoragev2' storage account. The top navigation bar includes the breadcrumb 'Home > packtadestoragev2' and the storage account name. A search bar contains 'alert'. To the right of the search bar, a red box highlights the '+ New alert rule' button, which has a blue plus sign icon and the text 'New alert rule'. The URL in the browser's address bar is 'https://portal.azure.com/#blade/HubsBlade/search/alert'. A 'Subscription : Azu' dropdown is visible at the bottom right.

Figure 2.29 – Adding a new alert

3. On the **Alerts | Create alert rule** page, observe that the storage account is listed by default in the **Resource** section. You can add multiple storage accounts in the same alert. Under the **Condition** section, click **Add condition**:

Create alert rule

Create an alert rule to identify and address issues when important conditions are found in your monitoring. When defining the alert rule, check that your inputs do not contain any sensitive content.

Scope

Select the target resource you wish to monitor.

Resource

 packtadestoragev2

[Edit resource](#)

Condition

Configure when the alert rule should trigger by selecting a signal and defining its logic.

Condition name

No condition selected yet

[Add condition](#)

Figure 2.30 – Adding a new alert condition

4. On the **Configure signal logic** page, select **Used capacity** under **Signal name**:

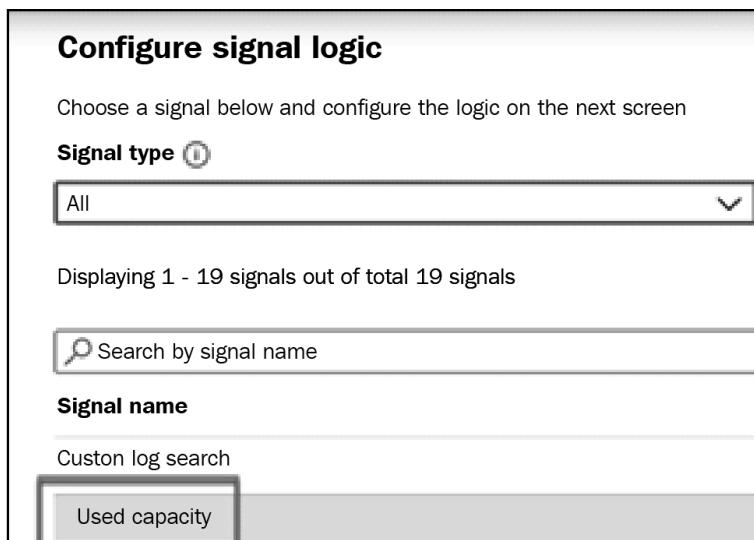


Figure 2.31 – Configuring the signal logic

5. On the **Configure signal logic** page, under **Alert logic**, set **Operator** to **Greater than**, **Aggregation type** to **Average**, and configure the threshold to **5 MiB**. We need to provide the value in bytes:

The screenshot shows the 'Configure signal logic' page with the following settings:

- Threshold**: Static (selected)
- Dynamic Thresholds** message: "Dynamic Thresholds is currently not available for this metric"
- Operator**: Greater than
- Aggregation type**: Average
- Threshold value**: 5
- Unit**: MiB
- Condition preview**: "Whenever the average used capacity is greater than 5 Mebibyte"
- Evaluated based on**: Aggregation granularity (Period): 1 hour; Frequency of evaluation: Every 1 Minute
- Done** button

Figure 2.32 – Configuring alert logic

Click **Done** to configure the trigger. The condition is added, and we'll be taken back to the **Create alert rule** page:

Scope

Select the target resource you wish to monitor.

Resource	Hierarchy
packtadestoragev2	Azure Pass - Sponsorship > PacktADE

[Edit resource](#)

Condition

Configure when the alert rule should trigger by selecting a signal and defining its logic.

Condition name	Time series monitored	Estimated monthly cost (USD)
Whenever the average used capacity is greater than 5 mebibyte	1	\$ 0.10
Add condition	1	Total \$ 0.10

i You can add up to 5 conditions with a static threshold for a metric alert rule. All conditions must be met for an alert to be triggered.

Figure 2.33 – Viewing a new alert condition

- The next step is to add an action to perform when the alert condition is reached. On the **Create alert rule** page, in the **ACTIONS GROUPS** section, click **Create**:

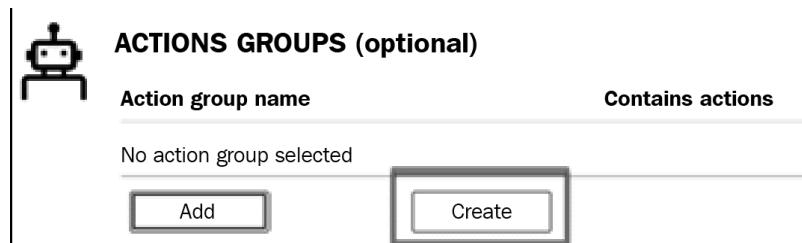


Figure 2.34 – Creating a new alert action group

- On the **Add action group** page, provide the **Action group name**, **Display name**, and **Resource group** details:

Home > packtadestoragev2 > Create an alert rule >

Create an action group ...

Basics Notifications Actions Tags Review + create

An action group invokes a defined set of notifications and actions when an alert is triggered. [Learn more](#)

Project details

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

Subscription *	①	Visual Studio Enterprise	✓	
Resource group *		①	packtadestorage	✓
Create new				

Instance details

Action group name *	①	packtadeactiongroup	✓
Display name *	①	packtad	✓
<small>This display name is limited to 12 characters</small>			

Figure 2.35 – Adding a new alert action group

8. In **Notifications**, provide an email address. Click on **Review + Create**:

The screenshot shows the 'Create an action group' interface. At the top, there are tabs for 'Basics', 'Notifications' (which is selected), 'Actions', 'Tags', and 'Review + create'. Below the tabs, there's a section titled 'Notifications' with the sub-instruction: 'Choose how to get notified when the action group is triggered. This step is optional.' Under 'Notification type', 'Email/SMS message/Push/Voice' is selected. The 'Name' field is set to 'Nagaraj'. A note below says 'Please configure the notification by clicking the edit button.' To the right, there's a 'Email/SMS message/Push/Voice' configuration panel. It has a heading 'Email/SMS message/Push/Voice' and a sub-instruction 'Add or edit an Email/SMS/Push/Voice action'. An 'Email' section is highlighted with a red box, containing a checked checkbox 'Email', an 'Email *' input field with the value 'navenkat@microsoft.com', and a 'Country code' dropdown set to '1'. Other sections include 'SMS' (unchecked), 'Azure app Push Notifications' (unchecked), and 'Voice' (unchecked). There's also a note about enabling the common alert schema with 'Learn more' and 'Yes'/'No' buttons. A large blue 'OK' button is at the bottom.

Figure 2.36 – Selecting the alert action

9. Click on **Create** to create the action group. We are then taken back to the **Create rule** page. The **Email** action is listed in the **Action Groups** section.
10. The next step is to define the **Severity**, **Alert rule name**, and **Alert rule description** details:

Home > packtadestoragev2 >

Create an alert rule

Scope Condition Actions Details Tags Review + create

Project details

Select the subscription and resource group in which to save the alert rule.

Subscription * ⓘ

Visual Studio Enterprise

Resource group * ⓘ

packtadestorage

[Create new](#)

Alert rule details

Severity * ⓘ

0 - Critical

Alert rule name * ⓘ

Alert when Usage size greater than 5 MIB

Alert rule description ⓘ

Alert when Usage size greater than 5 MIB

Enable upon creation ⓘ



Automatically resolve alerts ⓘ



Figure 2.37 – Creating an alert rule

11. Click the **Create alert rule** button to create the alert.

12. The next step is to trigger the alert. To do that, download `BigFile.csv` from the <https://github.com/PacktPublishing/Azure-Data-Engineering-Cookbook-2nd-edition/blob/main/Chapter2/BigFile.csv> file to the Azure storage account by following the steps mentioned in the *Creating containers and uploading files to Azure Blob storage using PowerShell* recipe of *Chapter 1, Creating and Managing Data in Azure Data Lake*. The triggered alerts are listed on the **Alerts** page, as shown in the following screenshot:

The screenshot shows the Azure Storage Account Alerts page for the resource group 'packtadestorage'. The top navigation bar includes 'Search (Ctrl+ /)', 'Create', 'Alert rules', 'Action groups', 'Alert processing rules (preview)', 'Refresh', and 'Feedback'. The main content area displays the following information:

- Subscription:** Visual Studio Enterprise
- Resource group:** packtadestorage
- Time range:** Past 24 hours
- Resource:** packtadestoragev2

Total alerts: 1 (Since 12/18/2021, 8:54 AM)

Smart groups (preview): 1 (0.00% Reduction)

Total alert rules: 1 (Enabled 1)

Severity	Total alerts	New	Acknowl
0 - Critical	1	1	0
1 - Error	0	0	0
2 - Warning	0	0	0
3 - Informational	0	0	0
4 - Verbose	0	0	0

Figure 2.38 – Viewing alerts

13. An email is sent to the email ID specified in the email action group. The email appears as shown in the following screenshot:

Azure: Activated Severity: 0 Alert when Usage size greater than 5 MIB



① If there are problems with how this message is displayed, click here to view it in a web browser.

⚠ Your Azure Monitor alert was triggered

Azure monitor alert rule Alert when Usage size greater than 5 MIB was triggered for packtadestoragev2 at December 18, 2021 18:10 UTC.

Alert rule description	Alert when Usage size greater than 5 MIB
Rule ID	/subscriptions/ /resourceGroups/packtadestorage/providers/microsoft.insights/metricAlerts/Alert when Usage size greater than 5 MIB View Rule >
Resource ID	/subscriptions/ /resourceGroups/packtadestorage/providers/Microsoft.Storage/storageAccounts/packtadestoragev2 View Resource >

Alert Activated Because:

Metric name	UsedCapacity
Metric namespace	storageAccounts/packtadestoragev2
Dimensions	AccountResourceId = /subscriptions/ /resourceGroups/packtadestorage/providers/Microsoft.Storage/storageAccounts/packta

Figure 2.39 – The alert email

How it works...

Setting up an alert is easy. At first, we need to define the alert condition (a trigger or signal). An alert condition defines the metrics and threshold that, when breached, trigger the alert. We can define more than one condition on multiple metrics for one alert.

We then need to define the action to be performed when the alert condition is reached. We can define more than one action for an alert. In our example, in addition to sending an email when the used capacity is more than 5 MB, we can configure Azure Automation to delete the old blobs/files in order to maintain the Azure storage capacity within 5 MB.

There are other signals, such as transactions, ingress, egress, availability, Success Server Latency, and Success E2E Latency, on which alerts can be defined. Detailed information on monitoring Azure storage is available at <https://docs.microsoft.com/en-us/azure/storage/common/storage-monitoring-diagnosing-troubleshooting>.

Securing an Azure storage account with SAS using PowerShell

A **Shared Access Signature (SAS)** provides more granular access to blobs by specifying an expiry limit, specific permissions, and IPs.

Using an SAS, we can specify different permissions to users or applications on different blobs, based on the requirement. For example, if an application needs to read one file/blob from a container, instead of providing access to all the files in the container, we can use an SAS to provide read access on the required blob.

In this recipe, we'll learn to create and use an SAS to access blobs.

Getting ready

Before you start, go through the following steps:

1. Make sure you have an existing Azure storage account. If not, create one by following the *Provisioning an Azure storage account using PowerShell* recipe in *Chapter 1, Creating and Managing Data in Azure Data Lake*.
2. Make sure you have an existing Azure storage container. If not, create one by following the *Creating containers and uploading files to Azure Blob storage using PowerShell* recipe.
3. Make sure you have existing blobs/files in an Azure storage container. If not, you can upload blobs by following the previous recipe.

4. Log in to your Azure subscription in PowerShell. To log in, run the Connect-AzAccount command in a new PowerShell window and follow the instructions.

How to do it...

Let's begin by securing blobs using an SAS.

Securing blobs using an SAS

Perform the following steps:

1. Execute the following command in the PowerShell window to get the storage context:

```
$resourcegroup = "packtadestorage"

$storageaccount = "packtadestoragev2"

#get storage context

$storagecontext = (Get-AzStorageAccount
-ResourceGroupName $resourcegroup -Name $storageaccount).Context
```

2. Execute the following commands to get the SAS token for the logfile1.txt blob in the logfiles container with list and read permissions:

```
#set the token expiry time
$starttime = Get-Date
$endtime = $starttime.AddDays(1)
# get the SAS token into a variable
$sastoken = New-AzStorageBlobSASToken -Container
"logfiles" -Blob "logfile1.txt" -Permission lr -StartTime
$starttime -ExpiryTime $endtime -Context $storagecontext
# view the SAS token.
$sastoken
```

3. Execute the following commands to list the blob using the SAS token:

```
#get storage account context using the SAS token
$ctx = New-AzStorageContext -StorageAccountName
$storageaccount -SasToken $sastoken
```

```
#list the blob details
Get-AzStorageBlob -blob "logfile1.txt" -Container
"logfiles" -Context $ctx
```

You should get output as shown in the following screenshot:

```
PS C:\Users\rajacct> $resourcegroup = "packtadestorage"
PS C:\Users\rajacct> $storageaccount = "packtadestoragev2"
PS C:\Users\rajacct> $storagecontext = (Get-AzStorageAccount -ResourceGroupName $resourcegroup -Name $storageaccount).Context
PS C:\Users\rajacct> $starttime = Get-Date
PS C:\Users\rajacct> $endtime = $starttime.AddDays(1)
PS C:\Users\rajacct> $sastoken = New-AzStorageBlobSASToken -Container "logfiles" -Blob "logfile1.txt" -Permission lr -StartTime $starttime -ExpiryTime $endtime -Context $storagecontext
PS C:\Users\rajacct> $ctx = New-AzStorageContext -StorageAccountName $storageaccount -SasToken $sastoken
PS C:\Users\rajacct> Get-AzStorageBlob -blob "logfile1.txt" -Container "logfiles" -Context $ctx

AccountName: packtadestoragev2, ContainerName: logfiles
Name          BlobType   Length    Contenttype      LastModified      AccessTier SnapshotTime      IsDeleted  VersionId
----          ----Type   ----     ----           ----           ----           ----           ----        ----
logfile1.txt  BlockBlob  201      text/plain      2021-12-18 16:25:16Z Hot            False

PS C:\Users\rajacct>
```

Figure 2.40 – Listing blobs using an SAS

4. Execute the following command to write data to `logfile1.txt`. Ensure you have the `LogFile1.txt` file in the `C:\ADECookbook\Chapter1\ Logfiles\` folder in the machine you are running the script from:

```
Set-AzStorageBlobContent -File C:\ADECookbook\Chapter1\
Logfiles\Logfile1.txt -Container logfiles -Context $ctx
```

You should get output as shown in the following screenshot:

```
PS C:\Users\rajacct> Set-AzStorageBlobContent -File "C:\ADECookbook\Chapter1\Logfiles\logfile1.txt" -Container logfiles
-Context $ctx

Confirm
Are you sure to overwrite 'https://packtadestoragev2.blob.core.windows.net/logfiles/logfile1.txt'?
[Y] Yes [N] No [S] Suspend [?] Help (default is "Y"): Y
Set-AzStorageBlobContent : This request is not authorized to perform this operation using this permission. HTTP Status
code: 403 - HTTP Error Message: This request is not authorized to perform this operation using this
permission.
ErrorCode: AuthorizationPermissionMismatch
ErrorMessage: This request is not authorized to perform this operation using this permission.
RequestId:052429c2-001e-0034-5f8e-f411ad000000
Time:2021-12-19T04:11:29.7356581Z
At line:1 char:1
+ Set-AzStorageBlobContent -File "C:\ADECookbook\Chapter1\Logfiles\logf ...
+ CategoryInfo          : CloseError: (:) [Set-AzStorageBlobContent], StorageException
+ FullyQualifiedErrorId : StorageException,Microsoft.WindowsAzure.Commands.Storage.Blob.SetAzureBlobContentCommand
```

Figure 2.41 – Uploading a blob using an SAS

The write fails, as the SAS token was created with list and read access.

Securing a container with an SAS

Perform the following steps:

1. Execute the following command to create a container stored access policy:

```
$resourcegroup = "packtadestorage"

$storageaccount = "packtadestoragev2"

#get storage context
$storagecontext = (Get-AzStorageAccount
-ResourceGroupName $resourcegroup -Name $storageaccount) .
Context

$starttime = Get-Date
$endtime = $starttime.AddDays(1)
New-AzStorageContainerStoredAccessPolicy -Container
logfiles -Policy writepolicy -Permission lw -StartTime
$starttime -ExpiryTime $endtime -Context $storagecontext
```

2. Execute the following command to create the SAS token:

```
#get the SAS token
$sastoken = New-AzStorageContainerSASToken -Name logfiles
-Policy writepolicy -Context
```

3. Execute the following commands to list all the blobs in the container using the SAS token:

```
#get the storage context with SAS token
$cxt = New-AzStorageContext -StorageAccountName
$storageaccount -SasToken $sastoken
#list blobs using SAS token
Get-AzStorageBlob -Container logfiles -Context $cxt
```

How it works...

To generate a shared access token for a blob, use the `New-AzStorageBlobSASToken` command. We need to provide the blob name, container name, permission (l = list, r = read, and w = write), and storage context to generate an SAS token. We can additionally secure the token by providing IPs that can access the blob.

We then use the SAS token to get the storage context using the `New-AzStorageContext` command. We use the storage context to access the blobs using the `Get-AzStorageBlob` command. Note that we can only list and read blobs and can't write to them, as the SAS token doesn't have write permissions.

To generate a shared access token for a container, we first create an access policy for the container using the `New-AzStorageContainerStoredAccessPolicy` command. The access policy specifies the start and expiry time, permission, and IPs. We then generate the SAS token by passing the access policy name to the `New-AzStorageContainerSASToken` command.

We can now access the container and the blobs using the SAS token.

3

Building Data Ingestion Pipelines Using Azure Data Factory

Azure Data Factory is the data orchestration service in Azure. Using Azure Data Factory, you can build pipelines that are capable of reading data from multiple sources, transforming the data, and loading the data into data stores to be consumed by reporting applications such as Power BI. Azure Data Factory much like **SQL Server Integration Services (SSIS)** in an on-premises world, provides a code-free UI for developing, managing, and maintaining data engineering pipelines.

Azure Data Factory is the bread and butter for a data engineer and understanding its fundamentals is extremely essential in building efficient pipelines. By the end of the chapter, you will know how to provision a data factory account, copy data from an Azure SQL database to a data lake using copy activity, use control flow activities, move data from SQL Server to a data lake, and choose options to trigger a data factory pipeline.

In this chapter, we'll cover the following recipes:

- Provisioning Azure Data Factory
- Copying files to a database from a data lake using a control flow and copy activity
- Triggering a pipeline in Azure Data Factory
- Copying data from a SQL Server virtual machine to a data lake using the Copy data wizard

Technical requirements

For this chapter, you will need the following:

- A Microsoft Azure subscription
- PowerShell 7 and above
- Microsoft Azure PowerShell, and an additional PowerShell module that's required for managing Azure components

Provisioning Azure Data Factory

To get started with Azure Data Factory, you need to run an Azure Data Factory account. An Azure Data Factory account is comprised of the following key components:

- **Linked services:** A component that maintains the connection credentials to data sources. An example of this is a connection to a SQL database/text file.
- **Dataset:** The data that's obtained after connecting to the data source using a linked service. An example of this is a group of tables or files connected via a linked service.
- **Activity:** A task that will process the dataset. An example of this is a copy activity that moves the data from a flat file to a database.
- **Data flow:** These are specific tasks that perform data transformations on datasets. An example of this is pivoting or sorting a dataset while it is being moved from source to destination. This can be done by a data flow transformation task.
- **Integration runtime:** This is the Azure Data Factory engine that works behind the scenes and provides the compute and resources to run the activities or tasks.
- **Pipeline:** A single entity that combines all the aforementioned components to connect, process, transform, and ingest the data to the destination. A single pipeline may contain multiple linked services, datasets, activities, and data flows.

How to do it...

In this recipe, we will be provisioning an Azure Data Factory using the Azure portal. Follow these steps:

1. Log in to portal.azure.com, click on **Create a resource**, and search for **Data Factory**. Select **Data Factory** and click on **Create**. Provide the data factory name, the resource group name, and location as shown in the following screenshot:

Create Data Factory

Basics Git configuration Networking Advanced Tags Review + create

Project details

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

Subscription *	<input type="text" value="Visual Studio Enterprise"/>
Resource group *	<input type="text" value="(New) PacktADEADF"/> Create new

Instance details

Region *	<input type="text" value="East US"/>
Name *	<input type="text" value="PacktADEADF"/> 
Version *	<input type="text" value="V2 (Recommended)"/>

Figure 3.1 – Create Data Factory

- Click on **Next: Git configuration**. Git configuration allows you to configure integration with Azure DevOps or GitHub. Git integration helps you save data factory pipelines as **Azure Resource Manager (ARM)** templates and lets you perform **continuous integration and continuous deployment (CI/CD)**. For this recipe, we will choose **Configure Git later**.

[Home](#) > [Create a resource](#) > [Data Factory](#) >

Create Data Factory

Basics **Git configuration** Networking Advanced Tags Review + create

Azure Data Factory allows you to configure a Git repository with either Azure DevOps or GitHub. Git is a version control system that allows for easier change tracking and collaboration.

[Learn more about Git integration in Azure Data Factory](#)

Configure Git later 

Figure 3.2 – Git configuration

- As for the remaining tabs (**Networking/Advanced/Tags**), they can remain as is. Click **Review + create** to create the data factory.

How it works...

It is fairly straightforward to create a data factory instance. The **PacktADEADF** data factory that we created in this recipe will be used to hold several datasets, pipelines, and data sources that are to be created in the following recipes in this chapter.

Copying files to a database from a data lake using a control flow and copy activity

In this recipe, we will be building a pipeline that will copy a group of files in blob storage to Azure SQL Database, but only if the filenames contain today's date as a suffix. Follow these steps:

1. Get the list of files to be copied using the **Get Metadata** activity in the data factory.
2. Use the **Filter** activity to filter the file whose suffix is the current date.
3. Use the **ForEach** activity to loop through the files.
4. Use the **Copy** activity to load the file into Azure SQL Database.

Getting ready

Follow these steps:

1. Provision a data factory, as explained in the *Provisioning Azure Data Factory* recipe of this chapter.
2. Log in to Azure PowerShell using `Connect-AzAccount`.
3. Execute the following command to create a storage account and a container:

```
$storageaccountname="packtadeadfadl"
$resourcegroup="PacktADEADF"
$containername="dataloading"

New-AzStorageAccount -ResourceGroupName $resourcegroup
-Name $storageaccountname -SkuName Standard_LRS -Location
'East US' -Kind StorageV2

$storagecontext = (Get-AzStorageAccount -ResourceGroupName
$resourcegroup -Name $storageaccountname).Context;

New-AzStorageContainer -Name $containername -Context
$storagecontext
```

4. Execute the following script to create a SQL Server database in the same resource group:

```
$resourcegroup="PacktADEADF"
$serverName = "packtadeadfsql"
```

```
$adminSqlLogin = "sqladmin"  
$password = "SQLPwdW5!k"  
$startIp = "0.0.0.0"  
$endIp = "255.255.255.255"  
$databasename = "sample"  
  
$server = New-AzSqlServer -ResourceGroupName  
$resourcegroup -ServerName $serverName -Location "Eastus"  
-SqlAdministratorCredentials $(New-Object -TypeName  
System.Management.Automation.PSCredential -ArgumentList  
$adminSqlLogin, $(ConvertTo-SecureString -String  
$password -AsPlainText -Force))  
  
$serverFirewallRule = New-AzSqlServerFirewallRule  
-ResourceGroupName $resourcegroup -ServerName $serverName  
-FirewallRuleName "AllowedIPs" -StartIpAddress $startIp  
-EndIpAddress $endIp  
  
$database = New-AzSqlDatabase -ResourceGroupName  
$resourcegroup -ServerName $serverName -DatabaseName  
$databaseName -RequestedServiceObjectiveName "S0"
```

5. Download the orderdtls-20211118.csv, orderdtls-20211119.csv, and orderdtls-20211120.csv files from <https://github.com/PacktPublishing/Azure-Data-Engineering-Cookbook-2nd-edition/tree/main/Chapter3> and upload them to the **dataloading** container in the **packtadeadfadl** storage account using the Azure portal, as shown in the following screenshot:

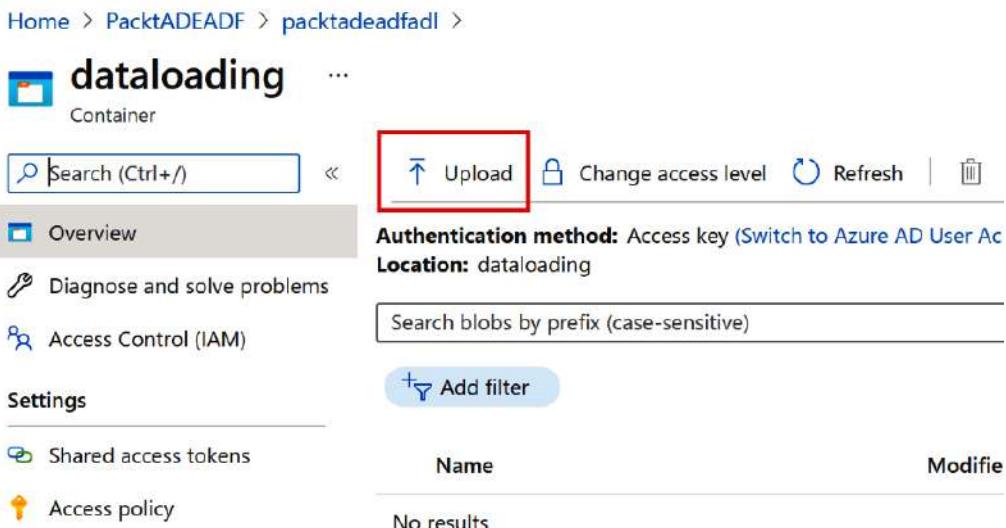


Figure 3.3 – Uploading the necessary files

- Select the files to be uploaded and hit the **Upload** button:

The screenshot shows the Azure Blob Storage interface for the 'dataloading' container. On the left, there's a sidebar with options like Overview, Diagnose and solve problems, Access Control (IAM), Settings, Shared access tokens, Access policy, Properties, Metadata, and Editor (preview). The main area displays a list of uploaded CSV files with columns: Name, Modified, Access tier, Archive status, and Blob type. The files listed are: orderdtls-20211118.csv, orderdtls-20211119.csv, orderdtls-20211120.csv, and orderdtls-20211121.csv. All files were modified on 8/25/2022, 8:22:05 AM, have a Hot (Inferred) access tier, and are Block blob type. To the right, there's an 'Upload blob' dialog box with tabs for 'Upload' and 'Advanced'. It shows the current uploads table with four entries, each with a status of 'Completed' and a size of 1 B.

Name	Modified	Access tier	Archive status	Blob type
orderdtls-20211118.csv	8/25/2022, 8:22:05 AM	Hot (Inferred)		Block blob
orderdtls-20211119.csv	8/25/2022, 8:22:05 AM	Hot (Inferred)		Block blob
orderdtls-20211120.csv	8/25/2022, 8:22:05 AM	Hot (Inferred)		Block blob
orderdtls-20211121.csv	8/25/2022, 8:22:05 AM	Hot (Inferred)		Block blob

Dismiss	Completed
orderdtls-20211121.csv	201 B / 201 B 1 B
orderdtls-20211120.csv	201 B / 201 B 1 B
orderdtls-20211119.csv	201 B / 201 B 1 B
orderdtls-20211118.csv	201 B / 201 B 1 B

Figure 3.4 – The files that have been uploaded

How to do it...

To copy the files in blob storage that have the current date as the suffix into Azure SQL Database, we will do the following:

- Create linked services to connect the blob storage and Azure SQL Database.
- Add the **Get Metadata** activity to get a list of files.
- Add the **Filter** activity to filter the files with the current date as the suffix.
- Add the **Copy** activity to copy the files that have been filtered to Azure SQL Database.

Creating a linked service

First, let's create two connections (linked services) – one for Azure SQL Database and another for blob storage:

- In the Azure portal, open the data factory that we provisioned. Click on **Open Azure Data Factory Studio**. Once the **Azure Data Factory Studio** opens, click on the **Manage** button:

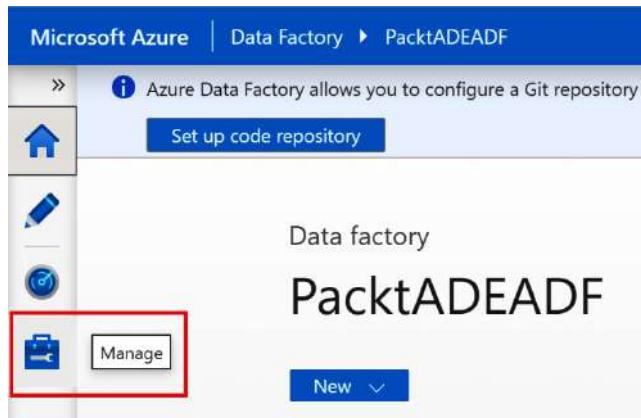


Figure 3.5 – Data Factory Studio – the Manage button

2. Click on **Linked Services**, then + New, and search for data under **Data store**. Select **Azure Data Lake Storage Gen2** and click **Continue**:

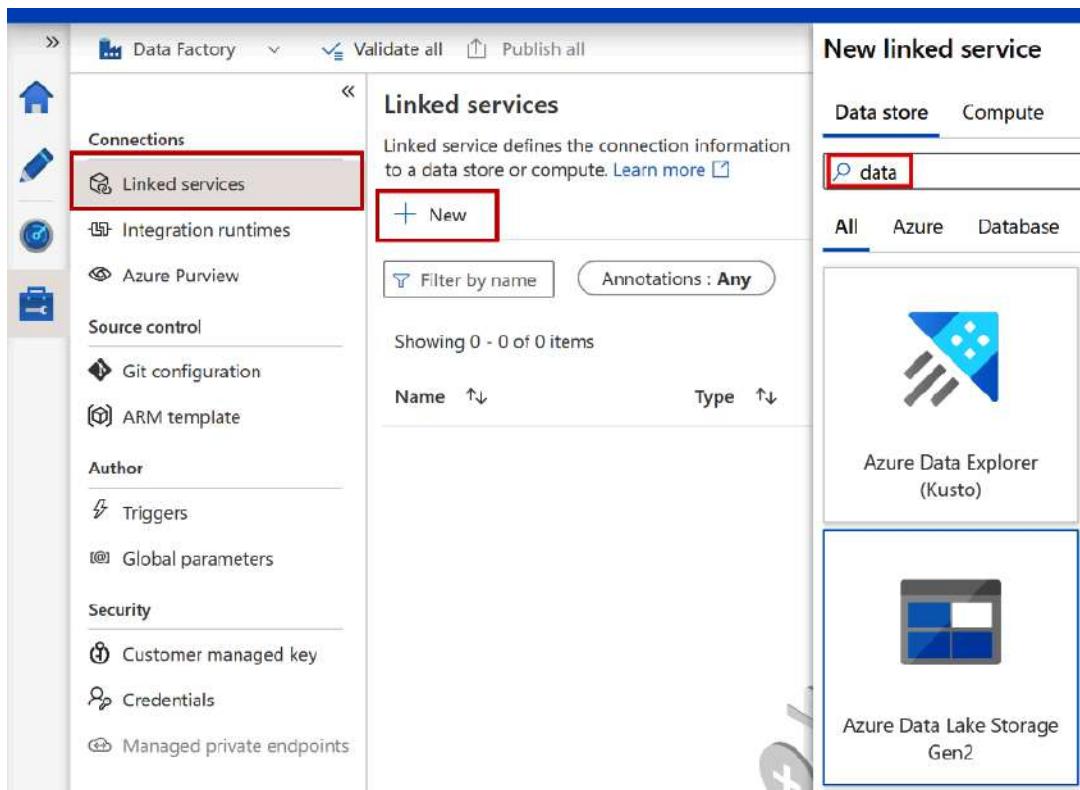


Figure 3.6 – Data Factory Studio – Data store

3. Create a connection to the storage account, as shown in the following screenshot:

New linked service (Azure Data Lake Storage Gen2)

Name *
DataLoading

Description

Connect via integration runtime * ⓘ
AutoResolveIntegrationRuntime

Authentication method
Account key

Account selection method ⓘ
 From Azure subscription Enter manually

Azure subscription ⓘ
Visual Studio Enterprise

Storage account name *
packtadeadfadl

Test connection ⓘ
 To linked service To file path

Annotations
+ New

> Parameters
> Advanced ⓘ

Figure 3.7 – New linked service (Azure Data Lake Storage Gen2)

4. Similarly, create a linked service for **Azure SQL Database**, as shown in the following screenshot. Set **User name** to `sqladmin` and **Password** to `SQLPwdW5!k`:

New linked service (Azure SQL Database)

Name *

Description

Connect via integration runtime * ⓘ

Account selection method ⓘ

From Azure subscription Enter manually

Azure subscription

Server name *



Database name *



Authentication type *



User name *

Password *

✓ Connection successful

Figure 3.8 – New linked service (Azure SQL Database)

Using the Get Metadata activity to get filenames

The first task is to get the list of files in the container. We'll do this using the **Get Metadata** activity. Follow these steps:

1. Create a new pipeline by clicking on the **Author** icon (the pencil-shaped icon on the left), then the + button, and then **Pipeline**:

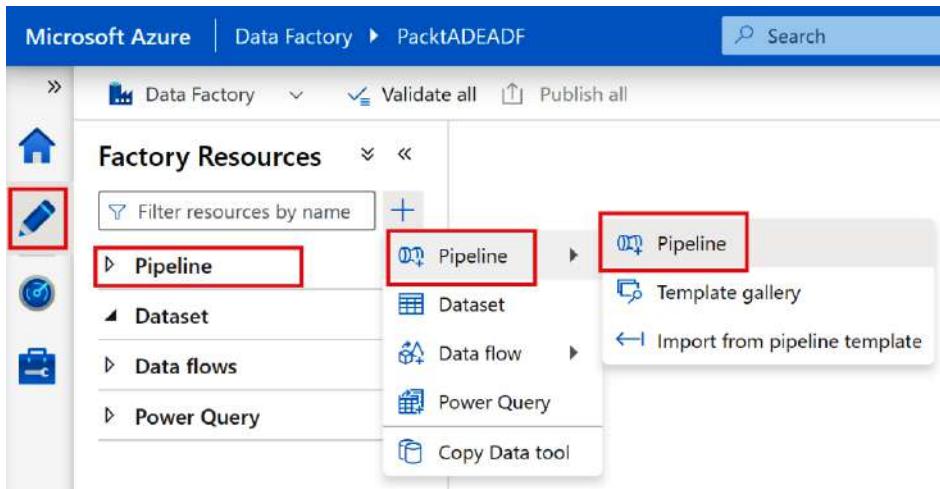


Figure 3.9 – Creating a pipeline

2. Under **Activities**, search for `get` and drag and drop the **Get Metadata** activity onto the pipeline:

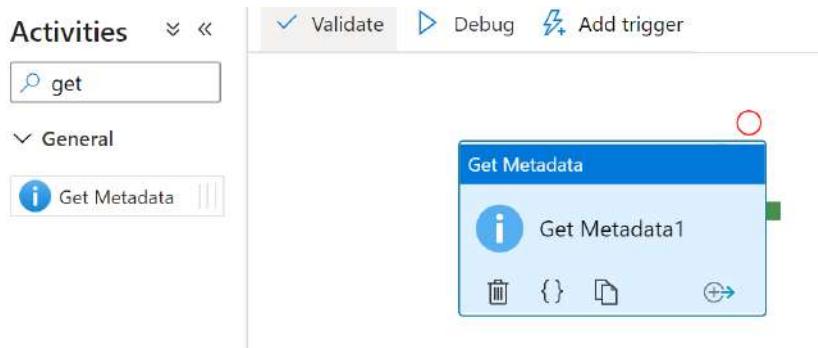


Figure 3.10 – Adding the Get Metadata activity

Set the name of the activity to `GetFilename`.

3. Under **Dataset**, click on the + New button to add a new dataset:



Figure 3.11 – Adding a dataset

4. Select **Azure Data Lake Storage Gen2** and select **CSV** as the file type. Name the dataset **OrderdtlsCSV**. Set **Linked service** to **DataLoading**, which we created earlier. Under **File path**, select the **dataloading** container. Then, check the **First row as header** box:

Set properties

Name
OrderdtlsCSV

Linked service *
DataLoading

File path
dataloading / Directory / File

First row as header

Import schema
 From connection/store From sample file None

> Advanced

Figure 3.12 – Set properties

5. Under **Field list**, click on the **+ New** button and select **Child items**:

The screenshot shows the 'Dataset' tab selected in the top navigation bar. Below it, a dropdown menu is set to 'OrderdtlsCSV'. A red box highlights the '+ New' button under the 'Field list' heading. Another red box highlights the 'Child items' option in the dropdown menu below.

Figure 3.13 – Field list

6. Hit the **Debug** button at the top and check the output. If no errors have been reported and the output shows the filenames, as shown in the following screenshot, then we have configured the **Get Metadata** task correctly and we can proceed to the next step:

The screenshot shows the 'Output' tab for a pipeline run. The pipeline run ID is 'c19f524a-d1c3-4bee-8c6e-e934e654c293'. A red box highlights the 'Get Metadata' task in the table, which has a duration of '00:00:12'. The 'Output' button at the bottom is also highlighted with a red box.

Name	Type	Run start	Duration
.GetFileName	Get Metadata	2021-11-21T01:00:33.010	00:00:12

Figure 3.14 – Checking the output

The following screenshot displays the output details:

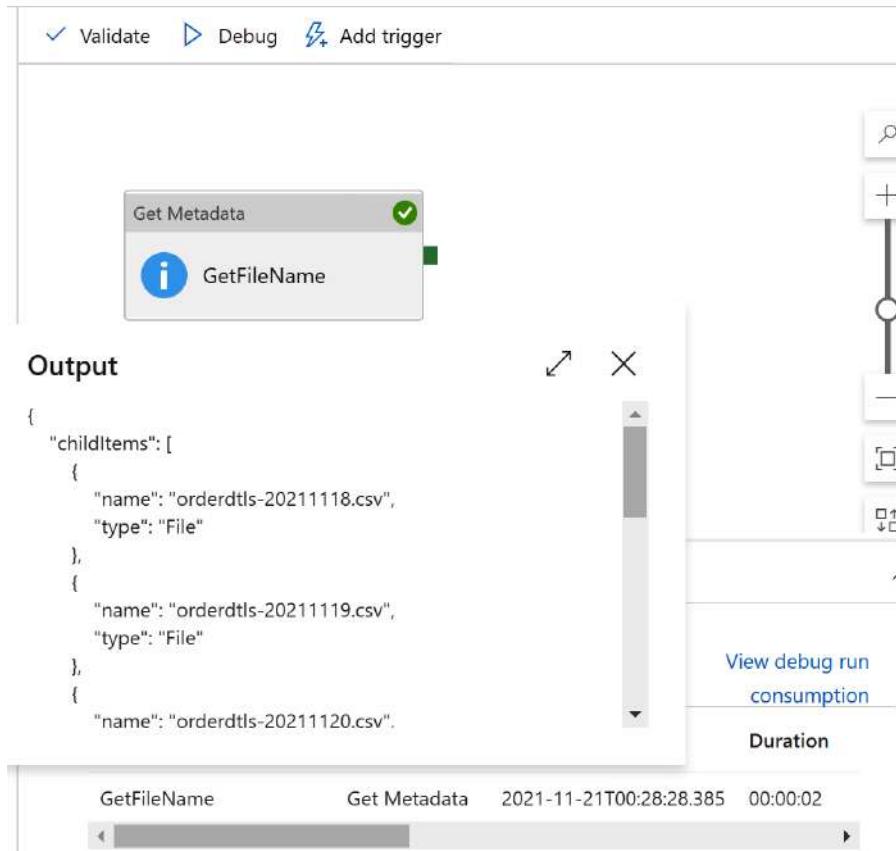


Figure 3.15 – Debug metadata

Filtering the current date files using the Filter activity

The next step is to filter the list of files that are returned by the **Get Metadata** activity for the files with the *current date* as the suffix. Let's add a **Filter** activity to the pipeline:

1. Search for **filter** in the **Activities** tab and drag and drop the activity onto the pipeline.
2. Connect the **Get Metadata** activity and the **Filter** activity.
3. Name the **Filter** activity **FilterToday'sDate**. Move to the **Settings** tab of the **Filter** activity.
4. For **Items**, click on the textbox and then click on **Add dynamic content** (**Alt + Shift + D**):

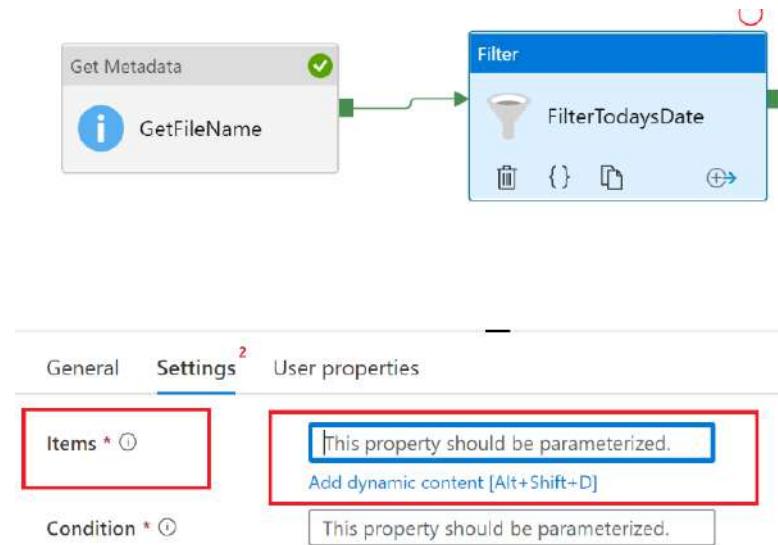


Figure 3.16 – Configuring the Filter activity

- Paste `@activity('GetFileName').output.childitems` into the **Items** field. This will retrieve the output array that was returned by the **Get Metadata** activity.
- Similarly, add `@endswith(item().name, concat('-', formatDateTime(utcnow(), 'yyyMMdd'), '.csv'))` for **Condition**, as shown in the following screenshot:

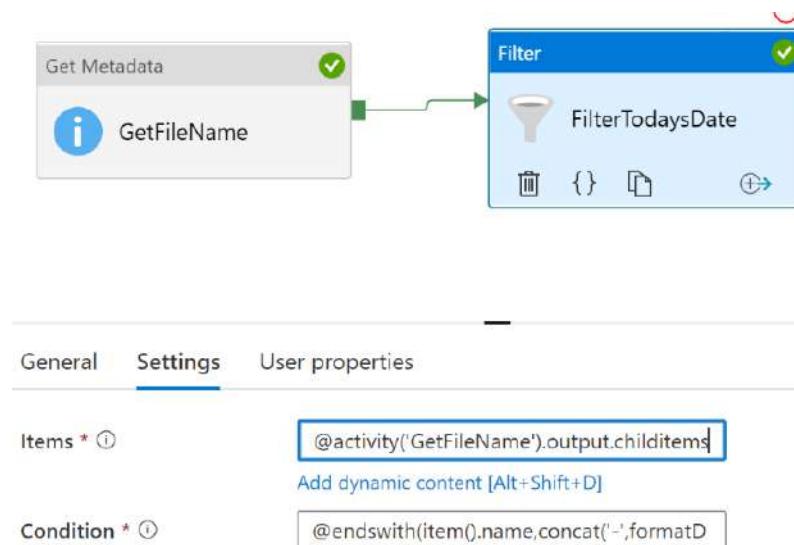


Figure 3.17 – Adding a condition to the Filter activity

7. Hit the **Debug** button to test this. Ensure that the **FilterTodaysDate** activity has been completed successfully and shows the filename with the current date as the suffix:

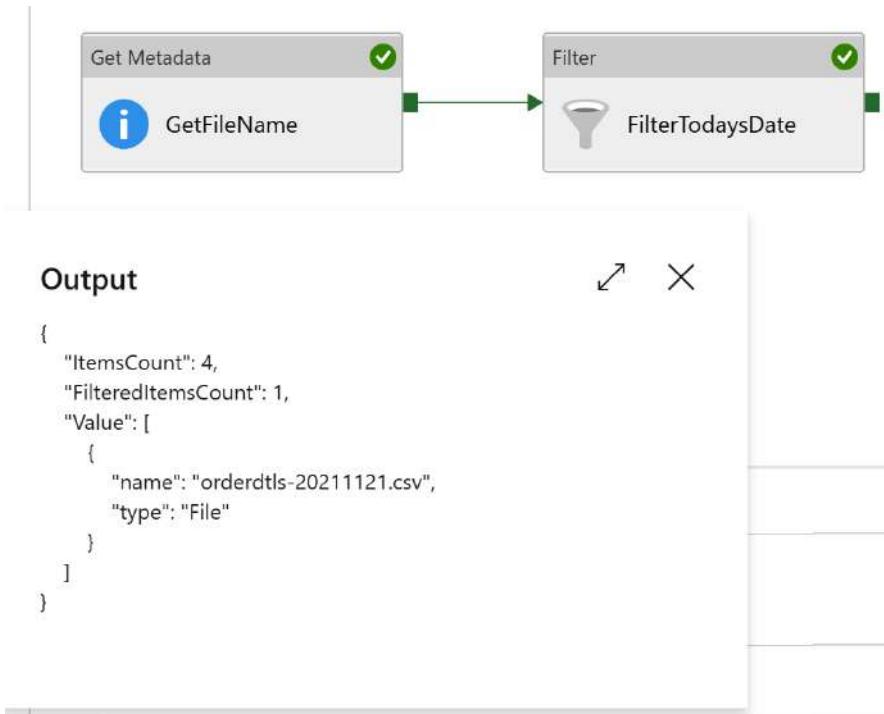


Figure 3.18 – Adding a condition to the Filter activity

Adding a ForEach activity to loop through the files

Now, we will create a **ForEach** activity to iterate through the files that are returned by the **Filter** activity. Follow these steps:

1. Search for **ForEach** under **Activities** and add the activity to the pipeline. Link it to the **FilterTodaysDate** activity.
2. Go to the **Settings** tab of the **ForEach** activity. For **Items**, click on **Add dynamic content** (**Alt + Shift + D**).
3. Under **Activity Outputs**, click on the **FilterTodaysDate** activity's output. This will automatically add `@activity('FilterTodaysDate').output`.
4. Append `.value` and set **Items** to `@activity('FilterTodaysDate').output.value`, as shown in the following screenshot. This will pass the filenames from the **FilterTodaysDate** activity to the **ForEach** activity:

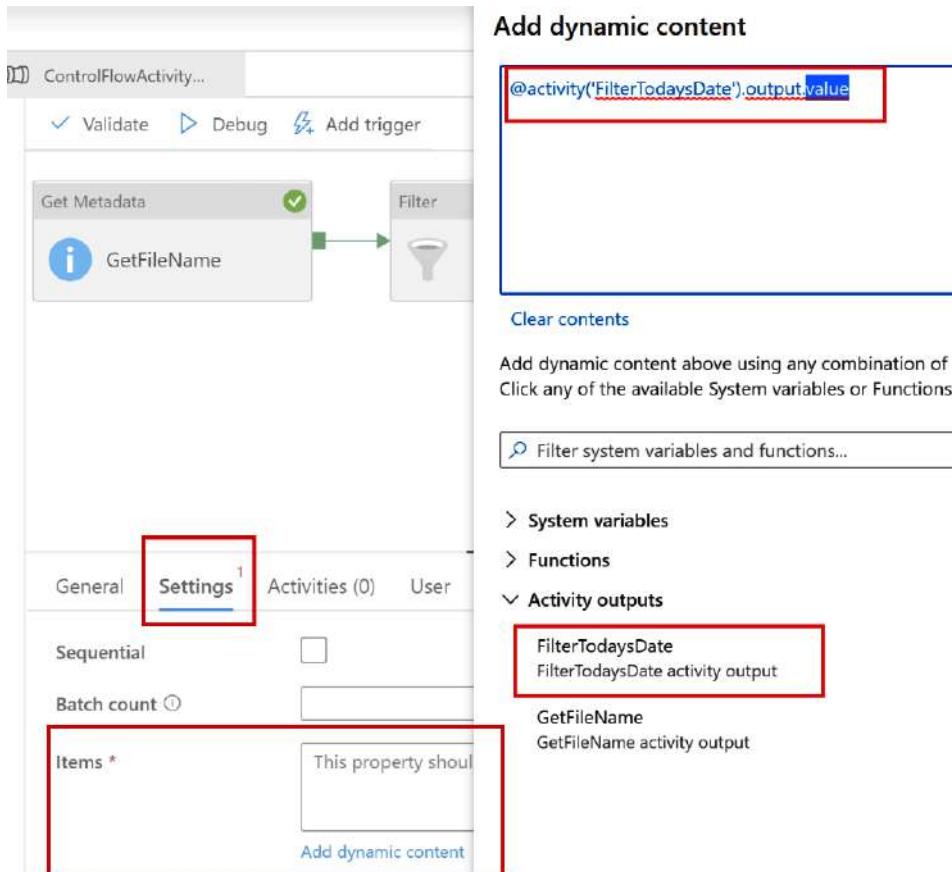


Figure 3.19 – Configuring items for the ForEach activity

5. Go to the **Activities (0)** tab and click on the pencil button.

Adding the Copy data activity to ingest files to Azure SQL Database

Finally, we will ingest files from the data lake to Azure SQL Database. We can do this by passing the files listed by the **ForEach** activity to the **Copy** activity. Follow these steps:

1. Search for **Copy Data** under **Activities** and add it to the pipeline.
2. Name the **Copy data** activity **CopyOrderDtltoSQL**.
3. Go to the **Source** tab. Select **OrderdtlsCSV** as the dataset since we need to copy CSV files to Azure SQL Database.
4. Under **File path type**, select **Wildcard file path**. In the last textbox, type `@item().name`, as shown in the following screenshot:

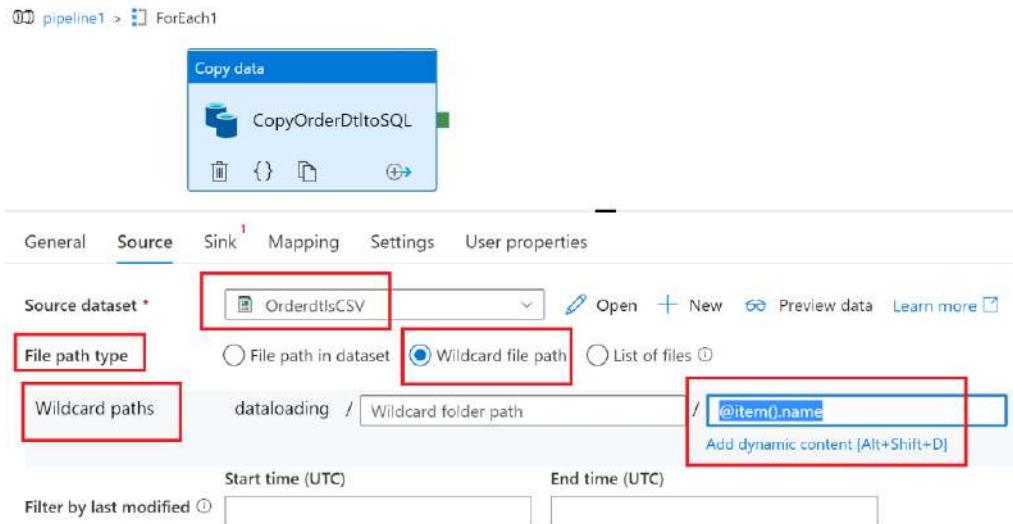


Figure 3.20 – Configuring the Copy data activity

5. Move to the **Sink** tab. Press **+ New** to add the dataset.
6. Search for **SQL** and select **Azure SQL Database**.
7. Select **SQLDB** as the linked service as we had created the connection to Azure SQL Database initially. Name the dataset as **OrderdtlSQL**. Click on the **Edit** checkbox below **Table name**. Provide table name as **dbo.orderdtls** as shown in the following screenshot. Select **None** option under **Import schema**. Press **OK**:

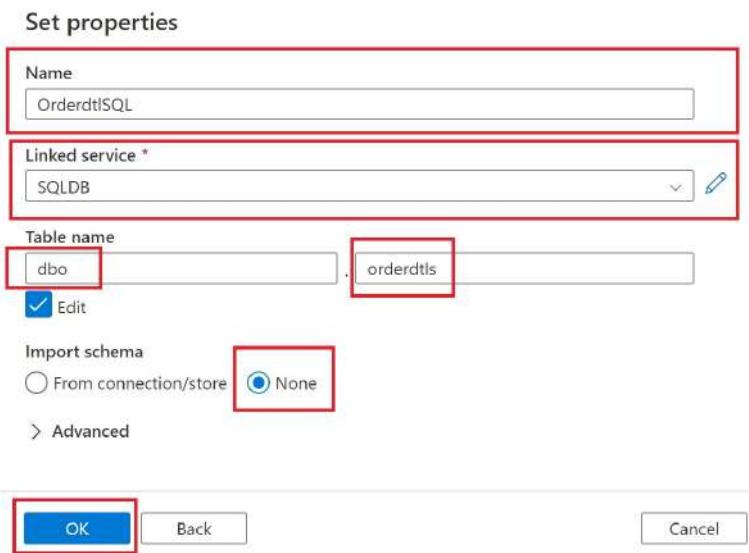


Figure 3.21 – Adding a SQL dataset

8. In the **Sink** tab, copy and paste the following script into the **Pre-copy script** field:

```
if not exists ( Select * from sys.objects where name like  
'orderdtls')  
Create table dbo.orderdtls(order_dt varchar(30),product  
varchar(100),cost int, quantity int, location  
varchar(100))
```

This will create a table called **orderdtls** for the first time and append rows to the same table in subsequent runs. All the other options can be left as is. Click on **pipeline1** to go back to the pipeline:

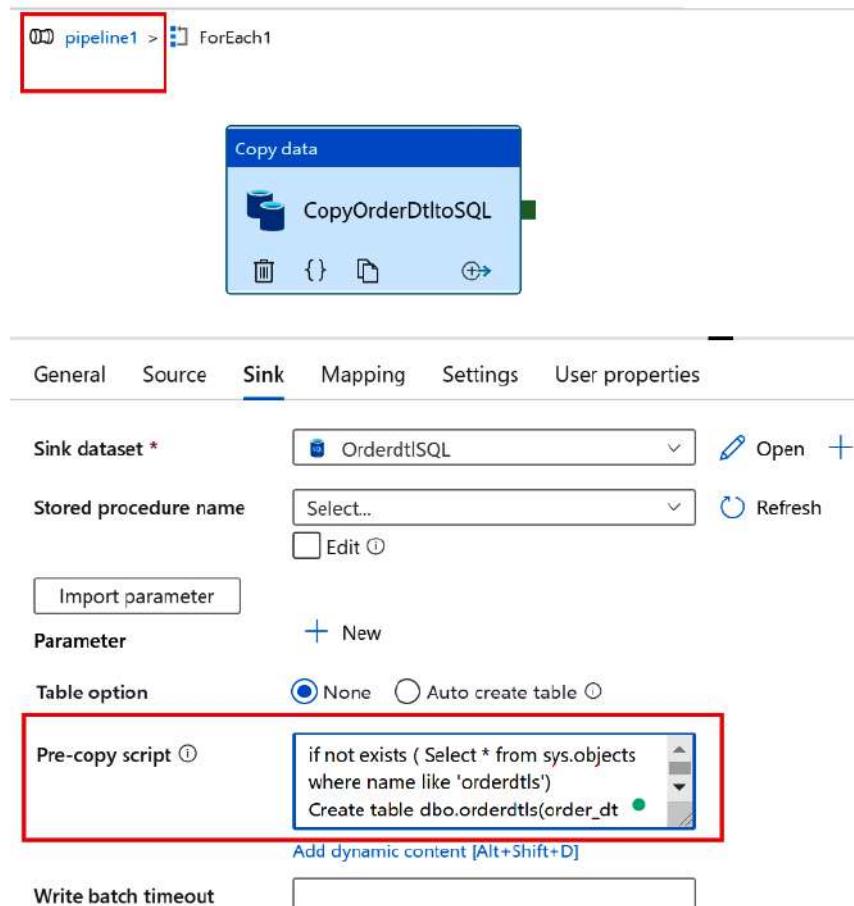


Figure 3.22 – Configuring Sink in the Copy data activity

9. Hit the **Debug** button to test all activities. The activities will complete, as shown in the following screenshot:

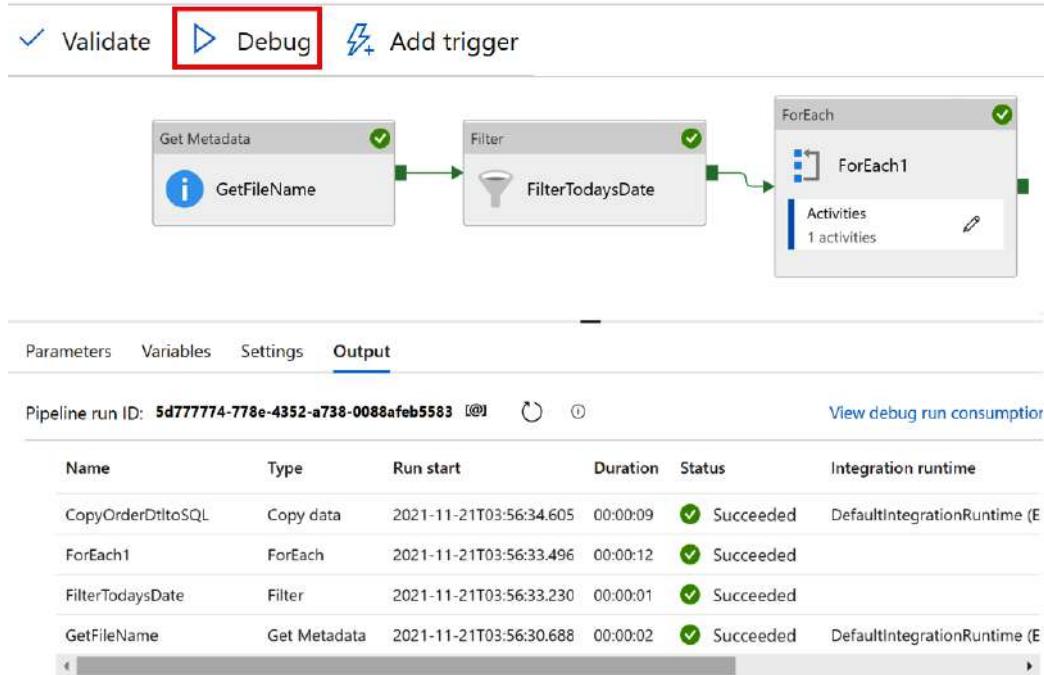


Figure 3.23 – Successful pipeline completion

10. Rename the pipeline `ControlFlowActivities` and hit the **Publish** button to save the pipeline. You can verify the result by querying Azure SQL Database from the Azure portal via **Query editor**, as shown in the following screenshot:

The screenshot shows the Azure SQL Database interface. On the left, the sidebar has a red box around the 'Query editor (preview)' option under the 'Power Platform' section. The main area shows a query editor titled 'Query 1' with the following content:

```
1 Select * from [dbo].[orderdtls]
```

The 'Run' button is highlighted with a red box. Below the query, the results are displayed in a table:

order_dt	product	cost	quantity	location
20211121	PC	1000	5	Singapore
20211121	keyboard	20	20	Dubai
20211121	cable	1	1000	Singapore
20211121	camera	50	50	Delhi
20211121	mobile	100	200	HongKong

Figure 3.24 – SQL results

How it works...

In this recipe, we performed four key steps to move the data from blob storage to Azure SQL Database:

1. We got the list of files in the container using the **Get Metadata** task.
2. We filtered for files while using the current date as a prefix using the **Filter** task.
3. We iterated through all the current date files using the **ForEach** task.
4. We ingested the file's content in a SQL table using the **Copy** task.

The following steps explain how this works:

1. In the **Get Metadata** task, the dataset is set to **OrderdtlsCSV**, which is linked to the **dataloading** container.
2. In the **Get Metadata** task, we set **Field list** to **Child items**, which means that the output of the **Get Metadata** task will be the filenames from the **dataloading** container. This is because the task's dataset is linked to that container:

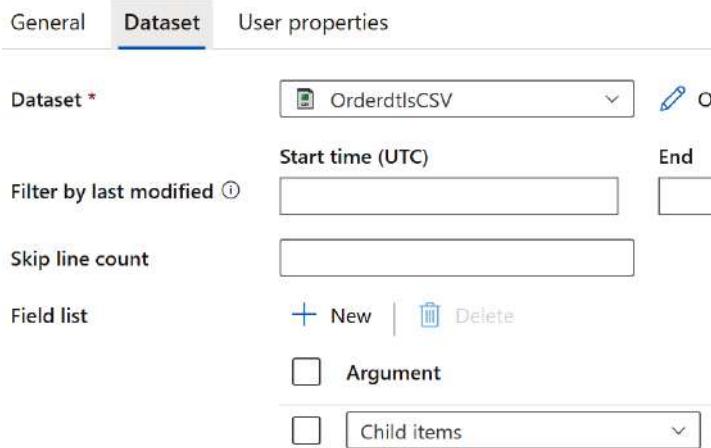


Figure 3.25 – Metadata child items

3. The **Filter** activity checks whether the filename of the **Get Metadata** activity has the current date as a suffix. In the **Filter** activity, two key settings are configured: **Items** and **Condition**. Let's look at how this works:
 - A. The **Filter** activity's **Items** parameter, which can be found under the **Settings** tab, accepts an array as input.
 - B. `@activity('GetFileName').output.childitems` is provided as input to **Items**, which means that all the filenames returned by the **Get Metadata** activity are passed as an array to the **Filter** activity.
 - C. The entries from the array that satisfy the condition specified in the **Condition** box will be returned as the output of the **Filter** activity.
4. For the **Condition** parameter, we provided the `@endswith(item().name, concat('-', formatDateTime(utcnow(), 'yyyMMdd'), '.csv'))` condition. Let's look at this in more detail:
 - `item().name`: This extracts each item from the array input, which will give us the filename
 - `formatDateTime(utcnow(), 'yyyMMdd')`: This converts the current date into `yyyymmdd` format
 - `concat('-', formatDateTime(utcnow(), 'yyyMMdd'), '.csv')`: This concatenates `-`, the current date in `yyyymmdd` format, and the `.csv` string
 - `endswith`: This checks for files (`item().name`) whose names end with the current date and with a `.csv` string as an extension:

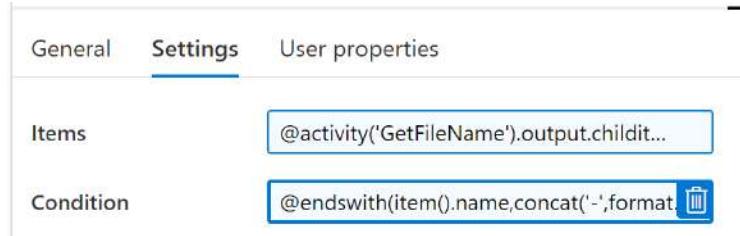


Figure 3.26 – The filename array and the date filter

5. The **ForEach** activity will perform a task for each item it receives from the **Filter** activity. In the **ForEach** activity, we received the filtered filenames as an array, which we then passed to the **Copy** activity.
6. We passed `@activity('FilterTodaysDate').output.value` for the **ForEach** activity's **Item** parameter. This ensures that the output of the **Filter** activity is passed as input to the **ForEach** activity:

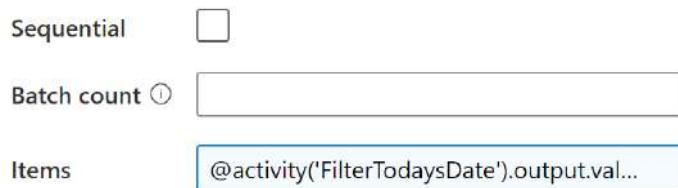


Figure 3.27 – The ForEach activity's array input

7. Under the **Activities** section of the **ForEach** activity, we added the **Copy** activity.
8. The **Copy** activity copies the files it received to Azure SQL Database. In the **Copy** activity, first, we set the source dataset to **OrderdtlsCSV** since it's linked to **dataloading**. We were unable to provide the filename as that would be dynamic. We had to set **File path type** to **Wildcard file path** since we need to provide the name dynamically. `@item().name` is passed as input in **Wildcard paths**, where `@item()` references the array item coming from the **ForEach** activity, and the name field, which is the name inside the array:

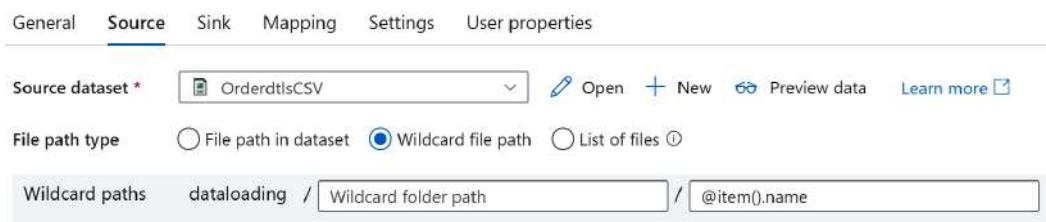


Figure 3.28 – The dynamic filename to copy

9. For the **Sink** tab, we created the dataset for Azure SQL Database. We set the table name to `dbo.orderdtls` using the **Edit** option, though the table doesn't exist in the database. The table will be created automatically in the first run. **Pre-copy script** has an `if exists` clause, which ensures a table is only created if it doesn't exist. Since the column names in the table and file match, no mapping needs to be performed during the transfer:

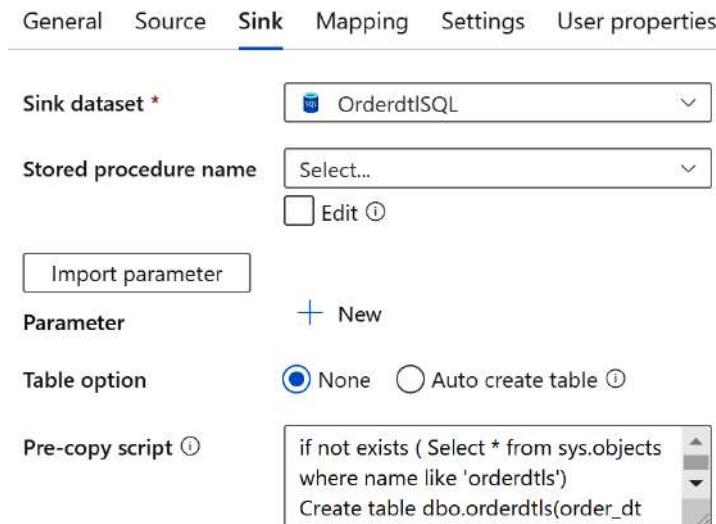


Figure 3.29 – Creating a table using Pre-copy script

By using various control activities, we can successfully transfer data from files to a database. Transferring files to a database is a common scenario in ETL workloads; you can use the preceding framework and customize the data transfer based on your requirements.

Triggering a pipeline in Azure Data Factory

An Azure Data Factory pipeline can be triggered manually, scheduled, or triggered by an event. In this recipe, we'll configure an event-based trigger to run the pipeline that we created in the previous recipe whenever a new file is uploaded to the Data Lake Store.

Getting ready

Before you start, perform the following steps:

- Log in to the Azure portal via PowerShell. To do this, execute the following command and follow the instructions to log in to Azure:

```
Connect-AzAccount
```

- Go to <https://portal.azure.com> and log in using your Azure credentials.
- Create the **ControlFlowActivities** pipeline, as specified in the previous recipe, if you haven't created it already.

How to do it...

To create the trigger, follow these steps:

1. The event trigger requires the `eventgrid` resource to be registered in the subscription. To do that, execute the following PowerShell command:

```
Register-AzResourceProvider -ProviderNamespace Microsoft.EventGrid
```

2. In the Azure portal, under All resources, open the **PacktADEADF** data factory that you created in the *Provisioning Azure Data Factory* recipe. On the **PacktADEADF** data factory overview page, select **Open Azure Data Factory Studio**. Click on the **Author** button on the left. Expand **Pipeline** and click on **ControlFlowActivities**, which was created in the previous recipe:

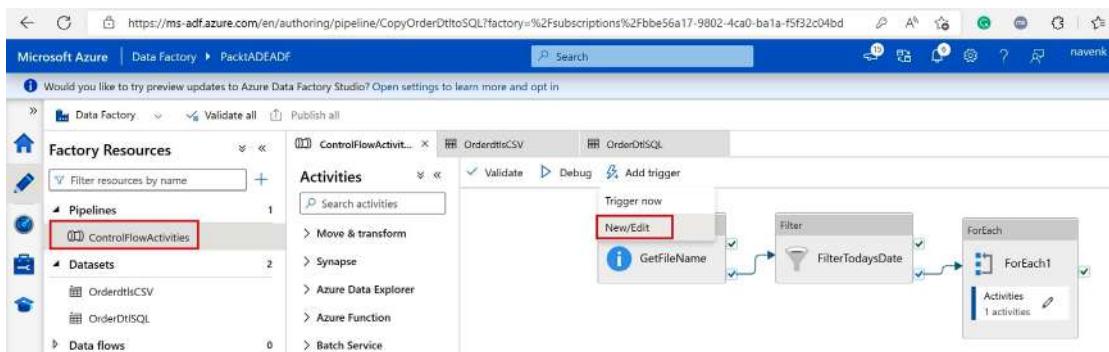


Figure 3.30 – Adding a trigger to a pipeline

3. Select **Add trigger** and then select **New/Edit**.

Note

To create **ControlFlowActivities**, please refer to the previous recipe.

4. In the **Add triggers** window, select **Choose trigger** and select **New**.
5. In the **New trigger** window, set **Name** as **NewFileTrigger**. Set the event's **Type** to **Storage event**. Use the **Storage account name** and **Container name** properties you created earlier. Under **Event**, select **blob Created**. This will ensure that any time a file is uploaded to the **dataloading** container, the pipeline will be triggered:

New trigger

Name *

NewFileTrigger

Description

Type *

Storage events

Account selection method * ⓘ

From Azure subscription Enter manually

Azure subscription ⓘ

Visual Studio Enterprise

Storage account name * ⓘ

packtadefadfl

Container name * ⓘ

dataloading

Blob path begins with ⓘ

Blob path ends with ⓘ

Event * ⓘ

Blob created

Blob deleted

Ignore empty blobs * ⓘ

Yes No

Figure 3.31 – Creating a trigger

6. Click **Continue** to create the trigger.
7. In the **Data preview** window, all the files in the **dataloading** container will be listed:

Event Trigger Filters

Container name: **dataloading**

Starts with:

Ends with:

11 blobs matched in "dataloading"

Refresh

SalesLTCustomer.txt
SalesLTCustomerAddress.txt
SalesLTProduct.txt
SalesLTProductCategory.txt
SalesSalesPerson.txt
SalesSalesReason.txt
orderdtls-20211118.csv
orderdtls-20211119.csv
orderdtls-20211120.csv
orderdtls-20211121.csv

1 - 11 of 11 items

< Previous Next > Go to

Figure 3.32 – The Data preview window

8. Click **Continue**.

9. In the **New trigger** window, we can specify the parameter values, if any, that are required by the pipeline to run:

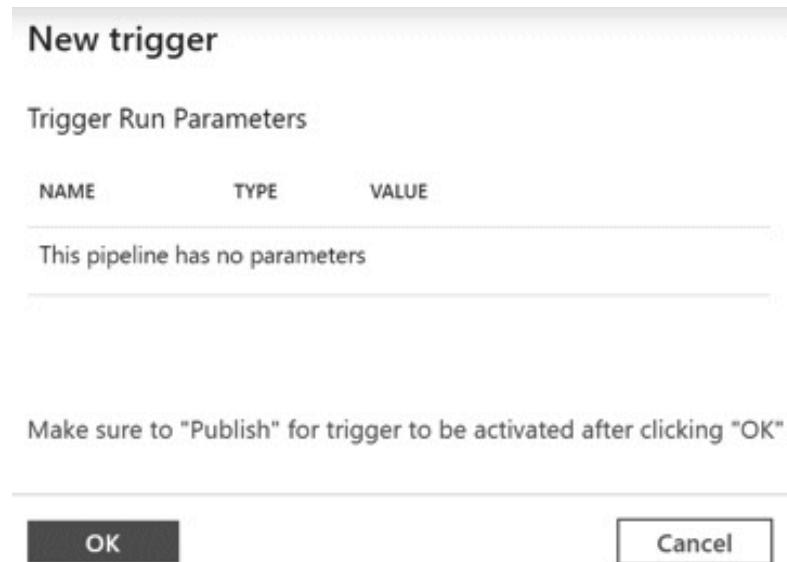


Figure 3.33 – Creating a new trigger

10. Click **OK** to create the trigger.
11. The trigger will be created. Click **Publish all** to save and apply these changes.
12. To see the trigger in action, do the following:
 - I. Download `orderdtls-Trigger.csv` from <https://github.com/PacktPublishing/Azure-Data-Engineering-Cookbook-2nd-edition/tree/main/Chapter3> to a local folder.
 - II. Log in to Azure PowerShell using `Connect-AzAccount`.
 - III. Execute the following command to upload the file to the **dataloading** container:

```
$storageaccountname = "packtadefad1"
$containername = "dataloading"
$resourcegroup = "PacktADEADF"
#Get the Azure Storage account context
$storagecontext = (Get-AzStorageAccount
-ResourceGroupName
$resourcegroup -Name $storageaccountname).Context;
```

```
Set-AzStorageBlobContent -File "C:\temp\orderdtls-
Trigger.csv" -Context $storagecontext -Blob orderdtls-
Trigger.csv -Container $containername
```

You will receive the following output:

```
PS C:\Users\navenkat> $storagecontext = (Get-AzStorageAccount -ResourceGroupName $resourcegroup -Name $storageaccountname).Context
PS C:\Users\navenkat> Set-AzStorageBlobContent -File "C:\temp\orderdtls-Trigger.csv" -Context $storagecontext -Blob orderdtls-Trigger.csv
Container $containername

AccountName: packtadeadfndl, ContainerName: dataloading
Name          BlobType   Length      ContentType           LastModified      AccessTier SnapshotTime
----          -----   ----       -----           -----           -----           -----
orderdtls-Trigger... BlockBlob 201      application/octet-stream 2022-08-25 02:50:03Z Hot

PS C:\Users\navenkat>
```

Figure 3.34 – File upload output

13. Once the file has been uploaded, **NewFileTrigger** will trigger the **ControlFlowActivities** pipeline.
14. To check the trigger and pipeline execution, open the **Monitor** window:

Pipeline name	Run start	Run end	Duration	Triggered by	Status
ControlFlowActivities	Aug 25, 2022, 10:58:02 am	Aug 25, 2022, 10:58:26 am	00:00:24	NewFileTrigger	Succeeded

Figure 3.35 – Viewing the trigger's status

You will see that the **ControlFlowActivities** pipeline was executed and that it was triggered by **NewFileTrigger**. This proves that the execution was triggered by the file being uploaded.

How it works...

Azure Event Grid, which we registered at the subscription level using the `Register-AzureResourceProvider` PowerShell command, helps track and trigger the pipeline when a file is uploaded to the data lake container. The storage event-based trigger makes it a powerful feature in data engineering projects, since pipelines need to be triggered when a file is uploaded by another batch process or other similar scenarios. You can add conditions such as blob prefix/suffix filters or parameters to trigger the pipeline, but only when specific files are loaded/deleted or when granular conditions must be met.

Copying data from a SQL Server virtual machine to a data lake using the Copy data wizard

A common scenario in data engineering projects is where you need to ingest data from a relational database engine such as SQL Server, Oracle, or MySQL to a data lake. This recipe will show you how to ingest data from SQL Server, which has been installed in an Azure VM, to an Azure Data Lake. This method will work in **on-premises** SQL Server to Azure Data Lake instances too, but you will need to install an integration runtime. This will be covered in the next chapter. In this recipe, we will focus on copying data from SQL Server in an Azure VM to a data lake. We will be using the user-friendly Copy data wizard to transfer the data.

Getting ready

Follow these steps:

1. Provision Azure Data Factory, as explained in the *Provisioning Azure Data Factory* recipe.
2. Log in to Azure PowerShell using the `Connect -AzAccount` command and provision the blob storage account and container, as shown in the following code block (only do this if you didn't do this in the previous recipe). Execute the following command to create a storage account and a container:

```
$storageaccountname="packtadefadl"
$resourcegroup="PacktADEADF"
$containername="dataloading"

New-AzStorageAccount -ResourceGroupName $resourcegroup
-Name $storageaccountname -SkuName Standard_LRS -Location
'East US' -Kind StorageV2

$storagecontext = (Get-AzStorageAccount -ResourceGroupName
$resourcegroup -Name $storageaccountname).Context;

New-AzStorageContainer -Name $containername -Context
$storagecontext
```

Provision the SQL Server VM by doing the following:

1. Log in to `portal.azure.com`.
2. Click on **Create a Resource**.
3. Search for **SQL Server**.
4. Select **SQL Server 2019 on Windows Server 2019**. Pick the **Free SQL Server License: SQL 2019 Developer on Windows Server 2019** option, as shown in the following screenshot:

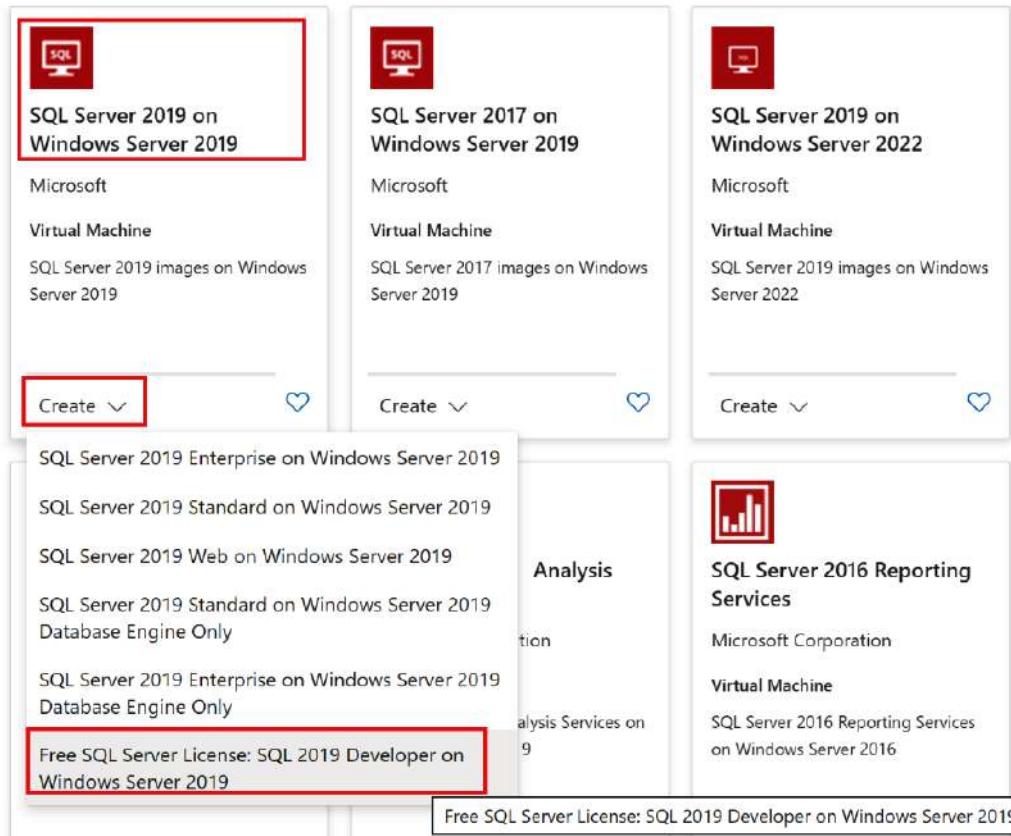


Figure 3.36 – Selecting the SQL Server 2019 image

5. Set **Resource group** to PacktADEADF and **Virtual machine name** to SQLVM. Then, set **Availability options** to **No infrastructure redundancy required**. After that, ensure that **Username** is set to sqladmin and that **Password** is set to SQLvmPwdW5!k. Leave **Select inbound ports** as is to allow the (RDP) 3389 port since it is allowed by default:

Home > Create a resource > Marketplace > SQL Server 2019 on Windows Server 2019 >

Create a virtual machine

Basics Disks Networking Management Advanced SQL Server settings Tags Review + create

Create a virtual machine that runs Linux or Windows. Select an image from Azure marketplace or use your own customized image. Complete the Basics tab then Review + create to provision a virtual machine with default parameters or review each tab for full customization. [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

Resource group * PacktADEADF Create new

Instance details

Virtual machine name * SQLVM

Region * (US) East US

Availability options * No infrastructure redundancy required

Security type Standard

Image * Free SQL Server License: SQL 2019 Developer on Windows Server 2019 - G See all images | Configure VM generation

VM architecture Arm64 x64
⚠️ Arm64 is not supported with the selected image.

Administrator account

Username * sqladmin

Password *

Confirm password *

Inbound port rules

Select which virtual machine network ports are accessible from the public internet. You can specify more limited or granular network access on the Networking tab.

Public inbound ports * None Allow selected ports

Select inbound ports * RDP (3389)

⚠️ This will allow all IP addresses to access your virtual machine. This is only recommended for testing. Use the Advanced controls on the Networking tab to create rules to limit inbound traffic to known IP addresses.

Figure 3.37 – Creating a SQL Server 2019 image

- Click on the **SQL Server settings** tab and set **SQL connectivity** to **Public (Internet)**. For **SQL Authentication**, choose **Enable**:

Home > Create a resource > Marketplace >

Create a virtual machine

Basics Disks Networking Management Advanced **SQL Server settings** Tags Review + create

Security & Networking

SQL connectivity * Public (Internet)

Port * 1433

SQL Authentication

SQL Authentication Disable Enable

Login name * sqladmin

Password *

Azure Key Vault integration Disable Enable

Figure 3.38 – Configuring the SQL Server 2019 network

7. Click on **Review + create**. It will take around 15 minutes to create the VM. Once the VM has been created, go to the VM's overview page in the Azure portal, get the **Public IP Address** information, and perform a remote desktop connection to the VM. Log in using your user ID and password – that is, `sqladmin/SQLvmPwdW5!k`.
8. Open **Windows PowerShell** in the SQL VM and run the following commands. These commands will create a folder and download the `adventureworks` backup file into the folder:

```
New-Item -Path c:\temp -ItemType directory
cd c:\temp
Invoke-WebRequest "https://github.com/
Microsoft/sql-server-samples/releases/download/
adventureworks/AdventureWorksLT2019.bak" -OutFile
"AdventureWorksLT2019.bak"
```

9. Open Command Prompt in the SQL VM and type the following:

```
Sqlcmd -e
```

10. Paste the following command in Command Prompt to restore the database to SQL Server:

```
RESTORE DATABASE [AdventureWorksLT2019] FROM DISK
= N'c:\temp\AdventureWorksLT2019.bak' WITH FILE =
1, MOVE N'AdventureWorksLT2012_Data' TO N'F:\data\
AdventureWorksLT2012.mdf', MOVE N'AdventureWorksLT2012_
Log' TO N'F:\log\AdventureWorksLT2012_log.
ldf', NOUNLOAD, STATS = 5
GO
```

11. Hit *Enter* on Command Prompt. This will restore the database, as shown in the following screenshot:

```
C:\Users\sqladmin>sqlcmd -e
:1) RESTORE DATABASE [AdventureWorksLT2019] FROM DISK = N'c:\temp\AdventureWorksLT2019.bak' WITH FILE = 1, MOVE N'AdventureWorksLT2012_Data' TO N'F:\data\AdventureWorksLT2012.mdf', MOVE N'AdventureWorksLT2012_Log' TO N'F:\log\AdventureWorksLT2012_log.ldf', NOUNLOAD, STATS = 5
2> GO
RESTORE DATABASE [AdventureWorksLT2019] FROM DISK = N'c:\temp\AdventureWorksLT2019.bak' WITH FILE = 1, MOVE N'AdventureWorksLT2012_Data' TO N'F:\data\AdventureWorksLT2012.mdf', MOVE N'AdventureWorksLT2012_Log' TO N'F:\log\AdventureWorksLT2012_log.ldf', NOUNLOAD, STATS = 5

15 percent processed.
30 percent processed.
46 percent processed.
61 percent processed.
77 percent processed.
92 percent processed.
100 percent processed.
Processed 840 pages for database 'AdventureWorksLT2019', file 'AdventureWorksLT2012_Data' on file 1.
Processed 2 pages for database 'AdventureWorksLT2019', file 'AdventureWorksLT2012_Log' on file 1.
RESTORE DATABASE successfully processed 842 pages in 0.059 seconds (111.427 MB/sec).
```

Figure 3.39 – Restoring the database

How to do it...

Now that the database has been restored, let's copy the data from the database into our data lake using the **Copy data** wizard in the data factory. Follow these steps:

1. Log in to portal.azure.com. Go to the data factory that you created earlier. Open **Azure Data Factory Studio**. Then, click on the **Ingest** button on the home page:

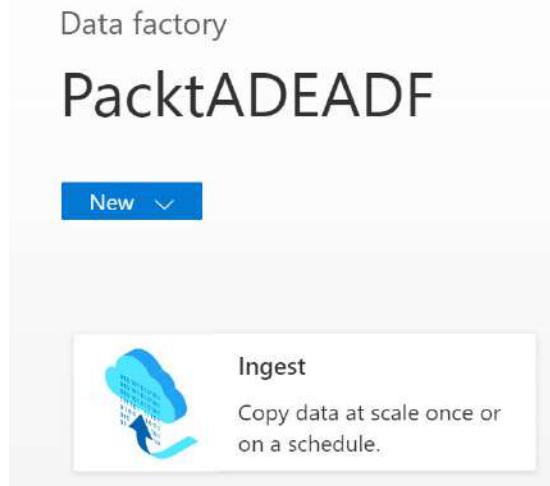
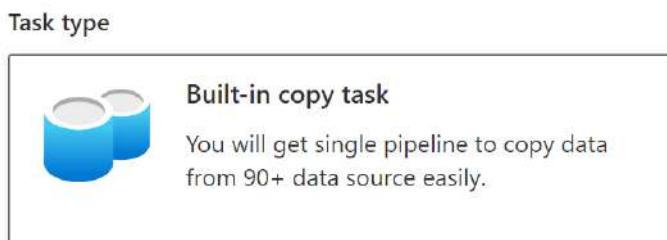


Figure 3.40 – Ingesting data

2. Select **Built-in copy task** and choose the **Run once now** option, as shown in the following screenshot:



You will get single pipeline to quickly copy objects from data

Task cadence or task schedule *

Run once now Schedule Tumbling window

Figure 3.41 – Built-in copy task

3. Set **Source type** to **SQL server** and click **+ New connection**, as shown in the following screenshot:

Source data store

Specify the source data store for the copy task. You can use an existing data store connection or specify a new data store.

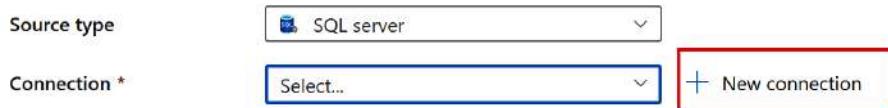


Figure 3.42 – New connection

4. Provide the SQL VM's public IP address under **Server name**. Set **Database name** as **AdventureWorksLT2019** and pick **SQL authentication** under **Authentication type**. Finally, set **User name** as **sqladmin** and **Password** as **SQLvmPwdW5 !k**. Then, click **Create**:

Figure 3.43 – Configuring the source dataset

5. Select a few tables you want to copy to the data lake:

The screenshot shows a list of tables from a database named 'SalesLT'. The tables listed are: dbo.BuildVersion, dbo.ErrorLog, SalesLT.Address, SalesLT.Customer (selected), SalesLT.CustomerAddress (selected), SalesLT.Product (selected), and SalesLT.ProductCategory (selected). The 'Existing tables' radio button is selected at the top. A filter bar at the top left allows 'Filter by name...'. A refresh button and status message 'Showing 12 out of 12 tables, 0 out of 3 views (4 selected)' are also present.

Figure 3.44 – Selecting tables

6. Hit **Next** twice to go to the **Destination data store** configuration screen. Select **Azure Data Lake Storage Gen2** under **Target type** and click **+ New connection**:

Destination data store

Specify the destination data store for the copy task. You can use an existing data store connection or specify a new data store.

The screenshot shows the 'Target type' dropdown set to 'Azure Data Lake Storage Gen2' and the 'Connection *' section with a 'Select...' dropdown and a '+ New connection' button. Both the 'Target type' dropdown and the '+ New connection' button are highlighted with red boxes.

Figure 3.45 – Destination connection

7. Pick the storage account you created earlier and click **Create** to create the connection:

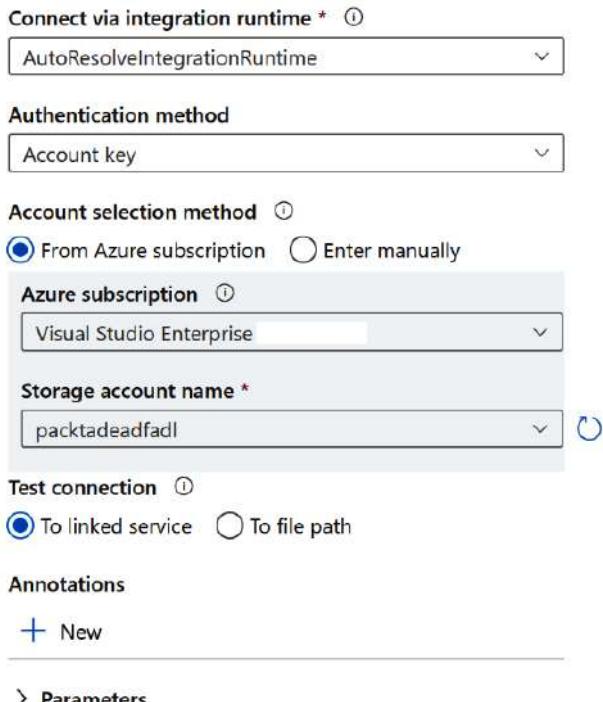


Figure 3.46 – Configuring the destination connection

- Under **Folder path**, click on **Browse**. Select the **dataloading** container you created earlier. Then, click **OK**:

Folder path

If the identity you use to access the data store only has permission to subdirectory instead of the entire account, specify the path to browse.

Browse

Select a file or folder.

Root folder

- dataloading**

Figure 3.47 – Browsing to the Root folder

9. Pick a **File format**. Check the **Add header to file** box to ensure that the column names are shown. Then, click **Next**:

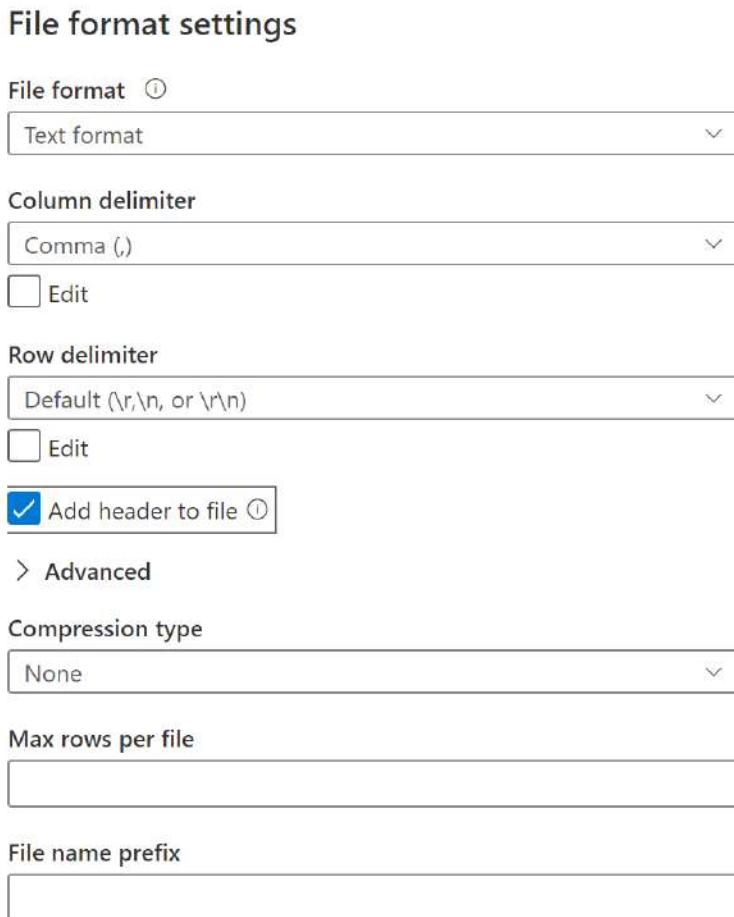


Figure 3.48 – File format settings

10. Set **Task name** as `CopySQLVMtoADL` and click **Next**:

Settings

Enter name and description for the copy data task, more options for data movement

Task name * `CopySQLVMtoADL`

Figure 3.49 – Using `CopySQLVMtoADL` as the task's name

11. Review your configuration on the **Summary** page and click **Next**. This will create the pipelines automatically and execute them. Click **Finish**:

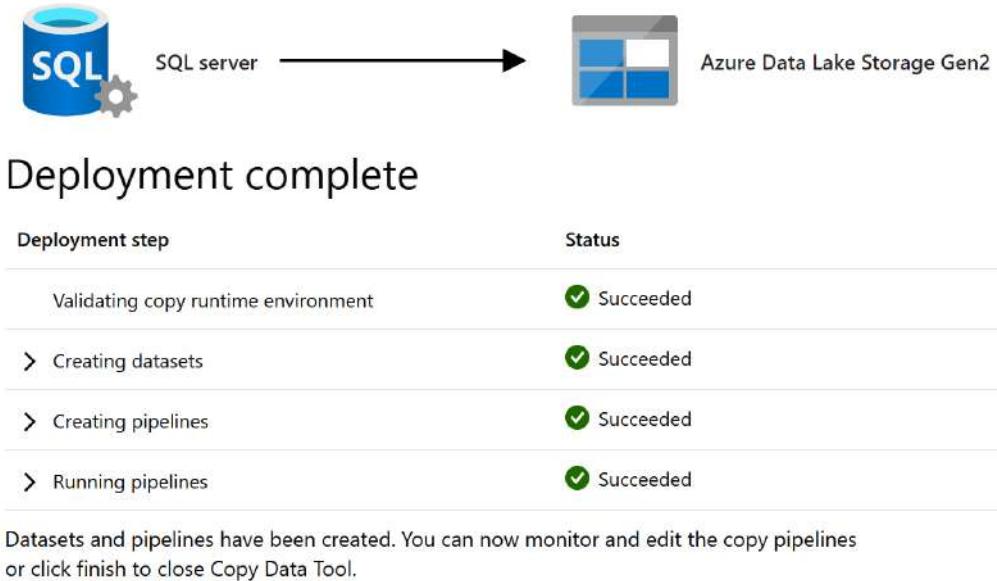


Figure 3.50 – The Summary page

12. Verify that the files have been created by checking the blob storage container in the Azure portal.

How it works...

Go to **Azure Data Factory Studio** and click on the **Author** button on the left. Expand **Pipeline** and notice that a new pipeline, **CopySQLVMtoADL**, has been created. Click on it. You will see that there's a **ForEach** activity in the pipeline. The **ForEach** activity is used to iterate through each table that needs to be copied:

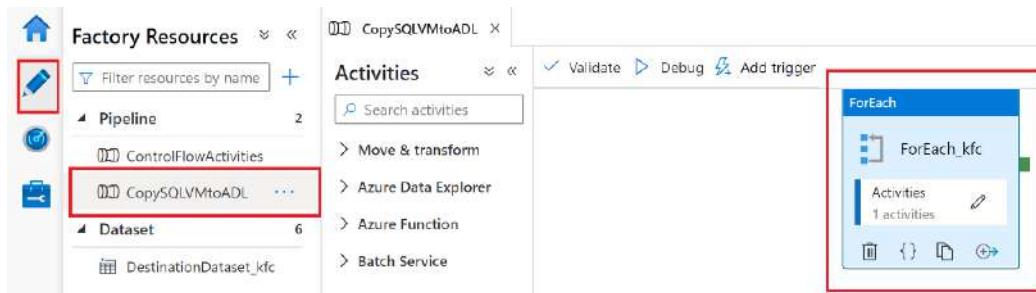


Figure 3.51 – The Copy data wizard pipeline

Click on **Activities**. You will notice a **Copy data** task whose source is **SQL Server** and the destination is **Azure Data Lake Storage Gen2**. The **Copy data** task will copy one table at a time from SQL Server to the data lake.

You will also notice that the source table name and destination filename come from the **ForEach** activity (`item().source.table /item().destination.fileName`). The Copy data wizard has automatically created the pipeline with the relevant activities to move the data:

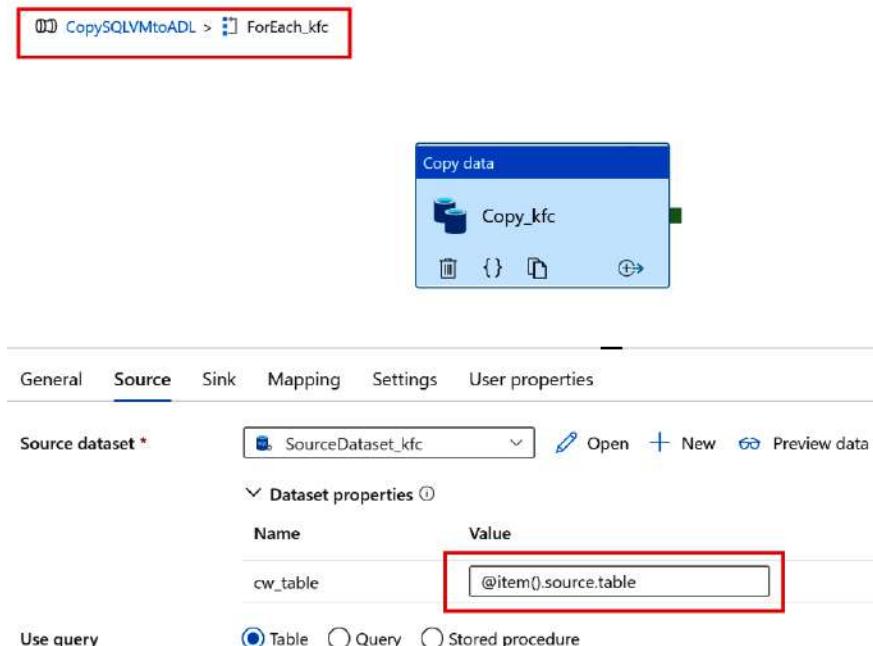


Figure 3.52 – The Copy data wizard

You can easily customize the pipeline or schedule it based on your needs to transfer data periodically. Copy data wizard saves so much time when it comes to configuring the datasets and the **ForEach** activity, which makes it easy to get started with data movement tasks.

Ensure that you delete the **PacktADEADF** resource group once you have finished since you will incur Azure consumption costs otherwise.

4

Azure Data Factory Integration Runtime

The Azure Data Factory **Integration Runtime (IR)** is the compute infrastructure that is responsible for executing data flows, pipeline activities, data movement, and **SQL Server Integration Services (SSIS)** packages. There are three types of IR: Azure, self-hosted, and Azure SSIS.

Azure IR is used to process data flows, data movement, and activities involving cloud-based data sources. By default, a type of Azure IR called **AutoResolveIntegrationRuntime** is created whenever a new data factory is created. AutoResolveIntegrationRuntime automatically determines the best location to run the IR. In addition to AutoResolveIntegrationRuntime, you can provision additional instances of Azure IR as well. In provisioned Azure IR instances, you may specify the preferred location to run the IR.

A self-hosted IR can be installed on-premises or on a virtual machine that uses the Windows OS and can be used to work with data on-premises or in the cloud. It can be used for data movement and activities.

The Azure SSIS IR is used to lift and shift existing SQL SSIS.

In this chapter, we'll learn how to use a self-hosted IR and Azure SSIS IR through the following recipes:

- Configuring a self-hosted IR
- Configuring a shared self-hosted IR
- Configuring high availability for a self-hosted IR
- Patching a self-hosted IR
- Migrating an SSIS package to Azure Data Factory

By the end of the chapter, you will know how to configure a self-hosted IR, share an IR across data factories, configure high availability for a self-hosted IR, upgrade a self-hosted IR, and migrate SSIS packages to Azure.

Technical requirements

For this chapter, the following are required:

- A Microsoft Azure subscription
- PowerShell 7
- Microsoft Azure PowerShell

Configuring a self-hosted IR

In this recipe, we'll learn how to configure a self-hosted IR and then use the IR to copy files from on-premises to Azure Storage using the **Copy data** activity.

Getting ready

To get started, do the following:

1. Log in to <https://portal.azure.com> using your Azure credentials.
2. Open a new PowerShell prompt and execute the following command to log into your Azure account from PowerShell:

Connect-AzAccount

3. You will need an existing Data Factory account. If you don't have one, create one by executing the following PowerShell script:

```
$resourceGroupName = "PacktADE";
$location = 'east us'
$dataFactoryName = "ADFPacktADE2";
$DataFactory = Set-AzDataFactoryV2 -ResourceGroupName
$resourceGroupName -Location $location -Name
$dataFactoryName
```

How to do it...

To configure a self-hosted IR, follow these steps:

1. In the Azure portal, open **Data Factory**, and then open the **Manage** tab:

The screenshot shows the Microsoft Azure portal interface for a Data Factory named 'ADFPacktADE2'. The left sidebar has a 'Connections' section with icons for Home, Data Factory, Validate all, Publish all, Connections, Linked services, Integration runtimes (which is highlighted with a red box), Azure Purview, Source control, Git configuration, and ARM template. Below the sidebar is a 'Integration runtimes' section with a heading 'The integration runtime (IR) is the compute infra...', a 'New' button (also highlighted with a red box), a 'Refresh' button, and a 'Filter by name' input field. It shows one item: 'AutoResolveIntegrationRuntime'. A red box also highlights the 'New' button.

Figure 4.1 – Opening the Manage tab under Data Factory

2. Select **New** and then select **Azure, Self-Hosted**:

Integration runtime setup

Integration Runtime is the native compute used to execute or dispatch activities. Choose what integration runtime to create based on required capabilities. [Learn more](#)

The screenshot shows the 'Integration runtime setup' page. It features two main options: 'Azure, Self-Hosted' (selected and highlighted with a red box) and 'Azure-SSIS'. The 'Azure, Self-Hosted' section includes an icon of a server in a cloud, the text 'Azure, Self-Hosted', and the description 'Perform data flows, data movement and dispatch activities to external compute.'. The 'Azure-SSIS' section includes an icon of a blue cylinder, the text 'Azure-SSIS', and the description 'Lift-and-shift existing SSIS packages to execute in Azure.'

Figure 4.2 – Selecting the Azure, Self-Hosted IR

3. Select **Azure, Self-Hosted** and click **Continue** to go to the next step. In the **Network environment** section, select **Self-Hosted**:

Integration runtime setup

Network environment:

Choose the network environment of the data source / destination or external compute to which the integration runtime will connect to for data flows, data movement or dispatch activities:

The screenshot shows the 'Network environment' setup page. It lists three options:

- Azure**: Represented by a blue cloud icon with a green plus sign. Description: "Use this for running data flows, data movement, external and pipeline activities in a fully managed, serverless compute in Azure."
- Self-Hosted**: Represented by a grey server icon with a green plus sign. Description: "Use this for running activities in an on-premises / private network". A "View more" link is below this description.
- Linked Self-Hosted**: Represented by a grey server icon with a green plus sign and a blue chain icon. Description: "Learn more".

A red box highlights the "Self-Hosted" option.

External Resources:

You can use an existing self-hosted integration runtime that exists in another resource. This way you can reuse your existing infrastructure where self-hosted integration runtime is setup.

The screenshot shows the "External Resources" section. At the bottom, there are three buttons: "Continue" (blue), "Back" (white), and "Cancel" (white).

Figure 4.3 – Selecting a network environment

4. Click **Continue** to go to the next step. In the next window, name the IR `selfhosted-onpremise` and click **Create**:

Integration runtime setup

Private network support is realized by installing integration runtime to machines in the same on-premises network/VNET as the resource the integration runtime is connecting to. Follow below steps to register and install integration runtime on your self-hosted machines.

The screenshot shows a configuration interface for creating an Integration Runtime (IR). It includes fields for Name, Description, and Type, along with action buttons at the bottom.

- Name:** A text input field containing "selfhosted-onpremise". This field is highlighted with a red rectangular border.
- Description:** A text area with placeholder text "Enter description here...".
- Type:** A text input field containing "Self-Hosted".
- Action Buttons:** At the bottom right, there are three buttons: "Create" (blue), "Back" (white), and "Cancel" (white).

Figure 4.4 – Creating the IR

5. The next step is to download and install the IR on the on-premises machine:

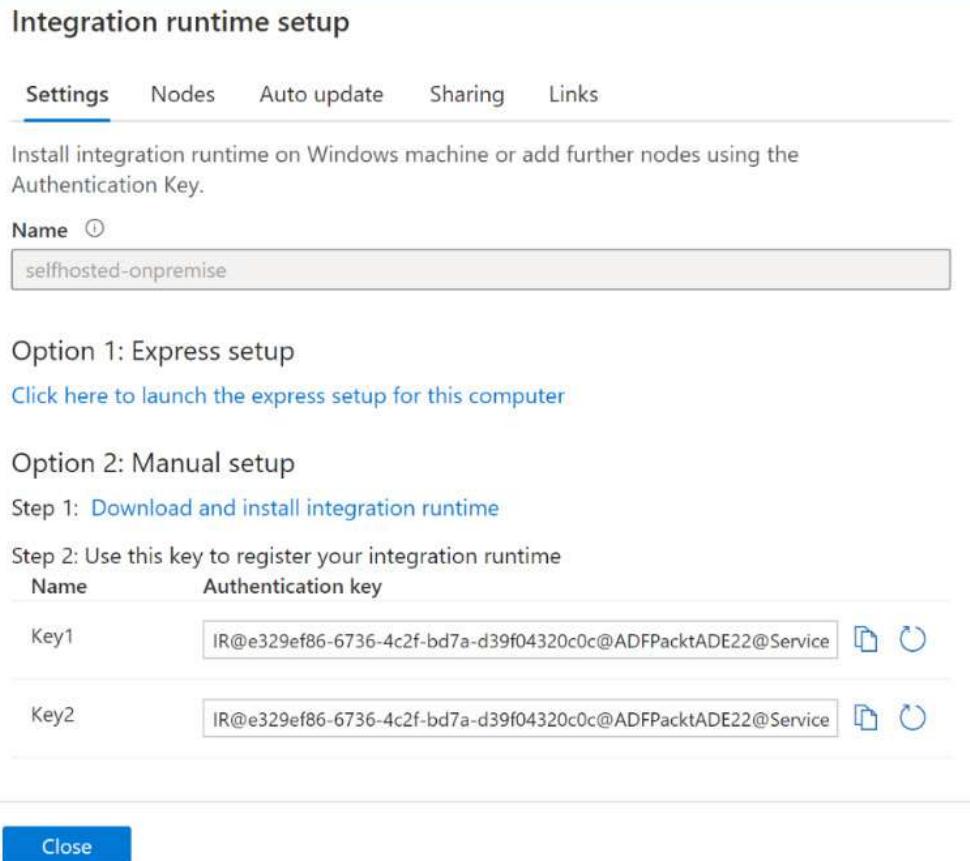


Figure 4.5 – Installing the IR

Note down the **Key1** value, as it's required in a later step.

6. If you are logged into the Azure portal from the machine where you wish to install the IR, select **Option 1: Express setup**. If you want to install it onto another machine, select **Option 2: Manual setup**.

We'll be using **Option 2: Manual setup**. Click on **Download and install integration runtime**. Download the latest version from the download center. Double-click the downloaded file and click **Run** to start the installation wizard. Follow the wizard to install the IR. When the installation is complete, copy and paste the authentication key, and click **Register**:

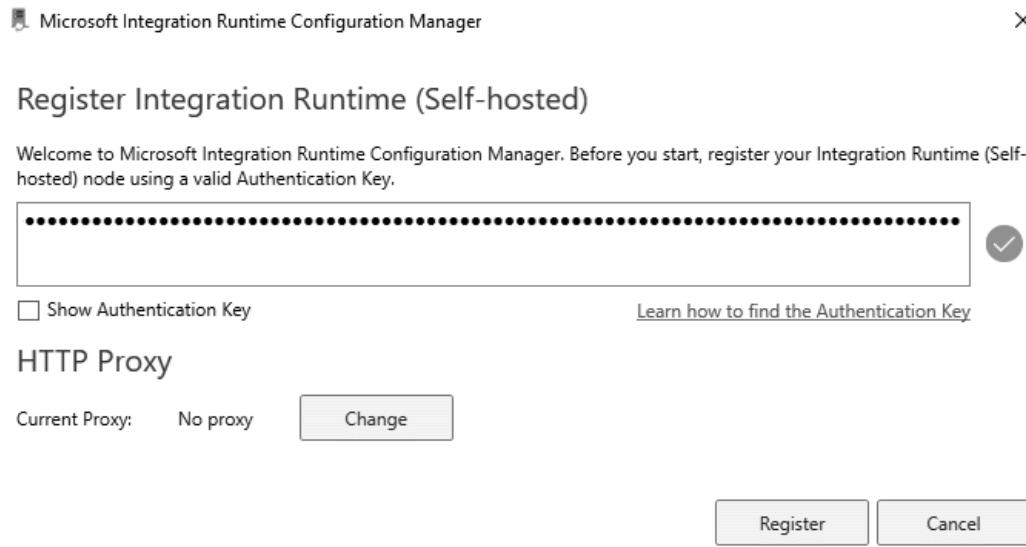


Figure 4.6 – Registering a self-hosted IR

- After the successful verification of the key, the computer or the node has now been registered as part of the `selfhosted-onpremise` IR. Hit the **Finish** button:

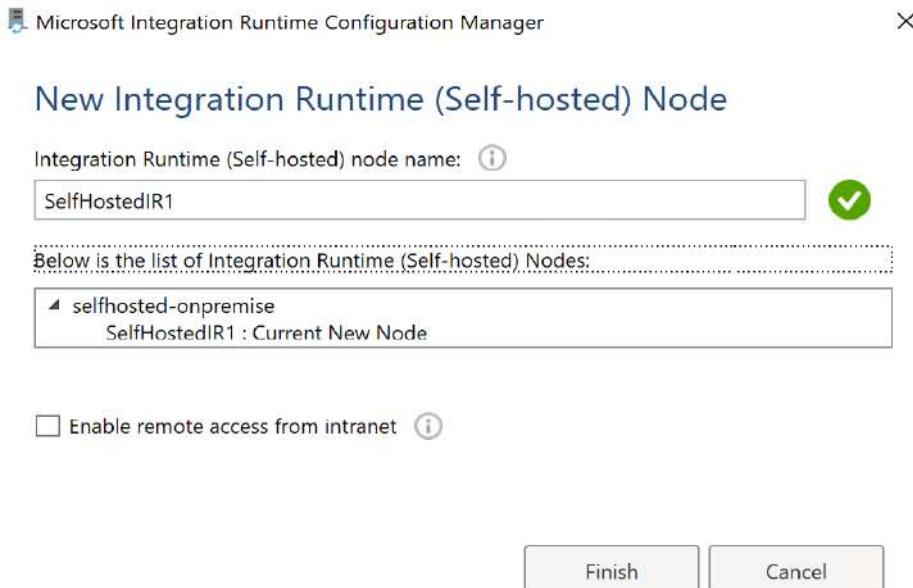


Figure 4.7 – Completing the self-hosted IR node configuration

8. Switch to the Azure portal and close the IR setup window. The new selfhosted-onpremise IR will be listed in the **Integration runtimes** window:

Integration runtimes				
Time zone : Chennai, Kolkata, Mumbai, New...				
All	Self-Hosted	Azure	Azure-SSIS	
Showing 1 - 2 of 2 items				
NAME ↑↓	TYPE ↑↓	SUB-TYPE ↑↓	STATUS ↑↓	REGION ↑↓
AutoResolveIntegrationRuntime	Azure	Public	<input checked="" type="checkbox"/> Running	Auto Resolve
selfhosted-onpremise	Self-Hosted	---	<input checked="" type="checkbox"/> Running	---

Figure 4.8 – Viewing the self-hosted IR status

9. Click on the **selfhosted-onpremise** IR to view the details:

The screenshot shows the Azure Data Factory Integration Runtimes blade. On the left, there's a navigation sidebar with options like Dashboards, Runs, Pipeline runs, Trigger runs, Runtimes & sessions, Integration runtimes (which is selected), and Data flow debug. The main area displays the details for the 'selfhosted-onpremise' runtime. It has tabs for Details (selected) and Activities, with Refresh and Edit buttons. The 'Details' section contains two tables. The first table shows basic information: STATUS (Running), TYPE (Self-Hosted), SUB-TYPE (---), and VERSION (5.19.8214.2). The second table shows metrics: HIGH AVAILABILITY ENABLED (True), LINKED COUNT (0), QUEUE LENGTH (0), and AVERAGE QUEUE DURATION (0.00s). Below these tables is a 'Node Details' section with a single row for 'SelfHostedIR1': Name (SelfHostedIR1), Status (Running), Version (5.19.8214.2), Available memory (4774MB), CPU utilization (0%), Network (In/Out) (0.32KBps/2.61KBps), and Concurrent jobs (0/6/6).

STATUS	TYPE	SUB-TYPE	VERSION
Running	Self-Hosted	---	5.19.8214.2

HIGH AVAILABILITY ENABLED	LINKED COUNT	QUEUE LENGTH	AVERAGE QUEUE DURATION
True	0	0	0.00s

Name	Status	Version	Available memory	CPU utilization	Network (In/Out)	Concurrent jobs (ru)
SelfHostedIR1	<input checked="" type="checkbox"/> Running	5.19.8214.2	4774MB	0%	0.32KBps/2.61KBps	0/6/6 More

Figure 4.9 – Viewing the self-hosted IR details

The **SelfHostedIR1** node is the name of the computer where we installed the IR. We'll now use the self-hosted IR to upload data from the files stored on-premises to Azure SQL Database. For this example, we will store the files in the self-hosted runtime machine (**SelfHostedIR1**). If the file needs to reside in a different machine other than that of the self-hosted IR, you can facilitate this by creating a file share and making it accessible from the self-hosted IR machine.

10. As the next step, let's prepare the source and destination for our file copy task, which is the file and Azure SQL Database. Download the `orderdtls.csv` file from <https://github.com/PacktPublishing/Azure-Data-Engineering-Cookbook-2nd-edition/tree/main/Chapter04>.
11. Upload to the IR machine (**SelfHostedIR1**) on a `C:\Chapter04\Data` path. We will be copying `orderdtls.csv` to **AzureSQLDatabase**.
12. Please execute the following PowerShell script to create a SQL database if you don't have Azure SQL Database to use as a destination. The following script creates a database called `sample` in the `packadesql` server:

```
$resourcegroup="PacktADE"
$serverName = "packadesql"
$adminSqlLogin = "sqladmin"
$password = "SQLPwdW5!k"
$startIp = "0.0.0.0"
$endIp = "255.255.255.255"
$databasename = "sample"
$server = New-AzSqlServer -ResourceGroupName
$resourcegroup -ServerName $serverName -Location "Eastus"
-SqlAdministratorCredentials $(New-Object -TypeName
System.Management.Automation.PSCredential -ArgumentList
$adminSqlLogin, $(ConvertTo-SecureString -String
$password -AsPlainText -Force))
$serverFirewallRule = New-AzSqlServerFirewallRule
-ResourceGroupName $resourcegroup -ServerName $serverName
-FirewallRuleName "AllowedIPs" -StartIpAddress $startIp
-EndIpAddress $endIp
$database = New-AzSqlDatabase -ResourceGroupName
$resourcegroup -ServerName $serverName -DatabaseName
$databaseName -RequestedServiceObjectiveName "S0"
```

13. Let's use the copy wizard to perform a quick data transfer:

- I. Go to the **Home** page of Azure Data Factory Studio and click the **Ingest** button:

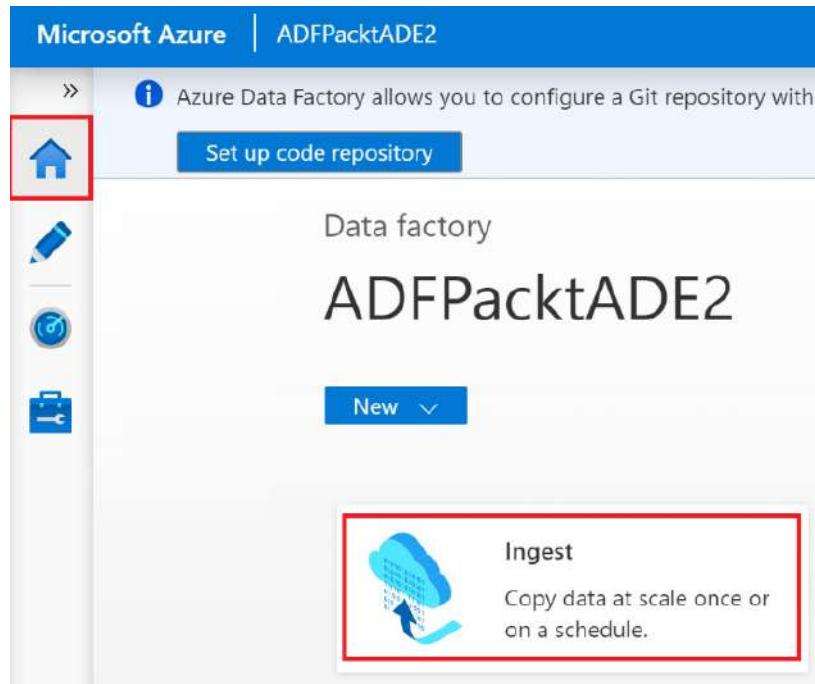


Figure 4.10 – Starting the copy wizard

- II. On **New connection**, search for **File**, as shown in the following screenshot:

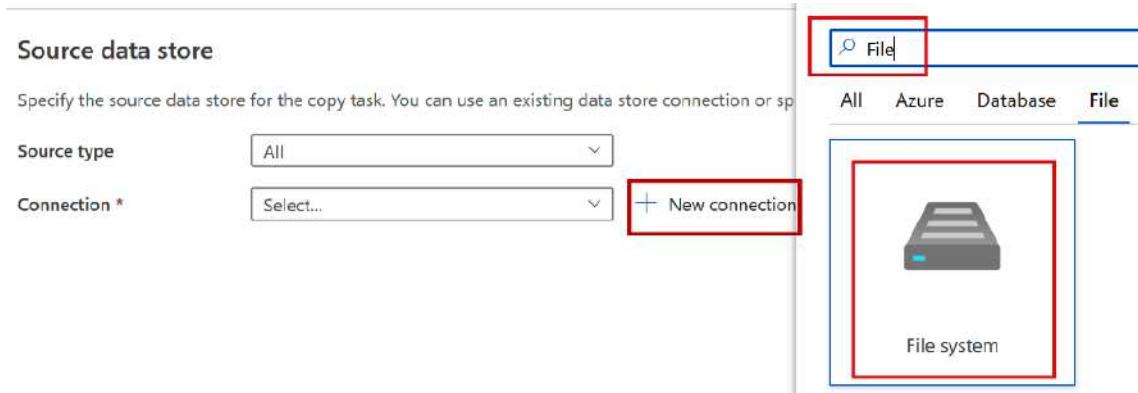


Figure 4.11 – Starting the copy wizard

III. Provide the IR machine credentials and path to the file, as shown in the following screenshot:

New connection (File system)

Name *

FileConnection

Description

Connect via integration runtime * ⓘ

selfhosted-onpremise



⚠ The credentials are stored in the machines of self-hosted integration runtime if you don't choose to store them in Azure Key Vault.

Host * ⓘ

c:\chapter04\data

User name *

packtadmin

Password

Azure Key Vault

Password *

.....

Figure 4.12 – Configure file connection

IV. Once the connection is created, you will be taken back to the **Source data store** page under **Copy Data tool**. Provide the file to be copied. Click on **Next**:

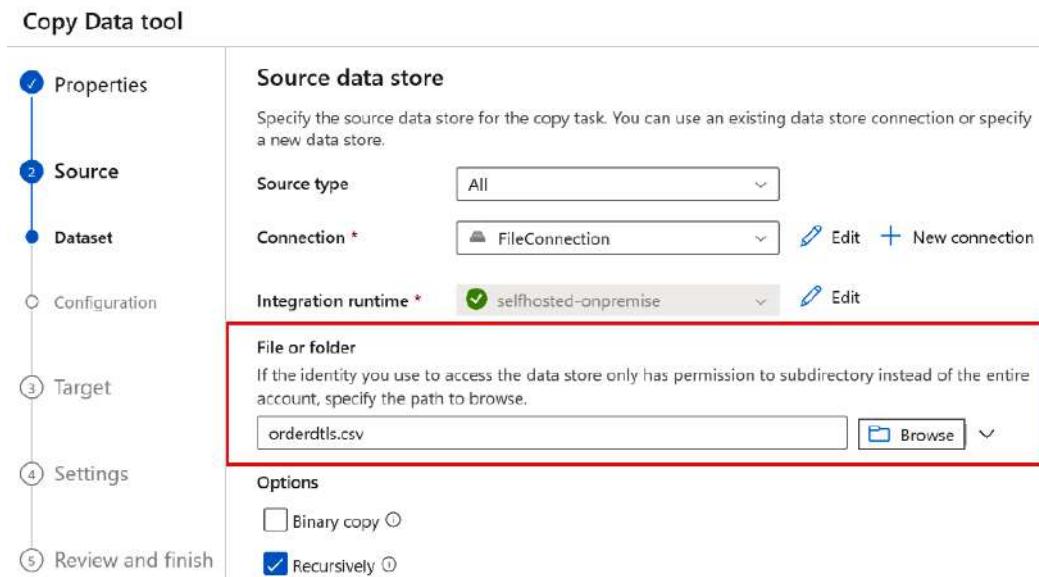


Figure 4.13 – File details

- V. The **Configuration** section in the **Source** section of **Copy Data tool** can remain with the defaults. On the **Target** page of **Copy Data tool**, click **+ New connection** and pick **Azure SQL Database** as the destination data store, as shown in the following screenshot:

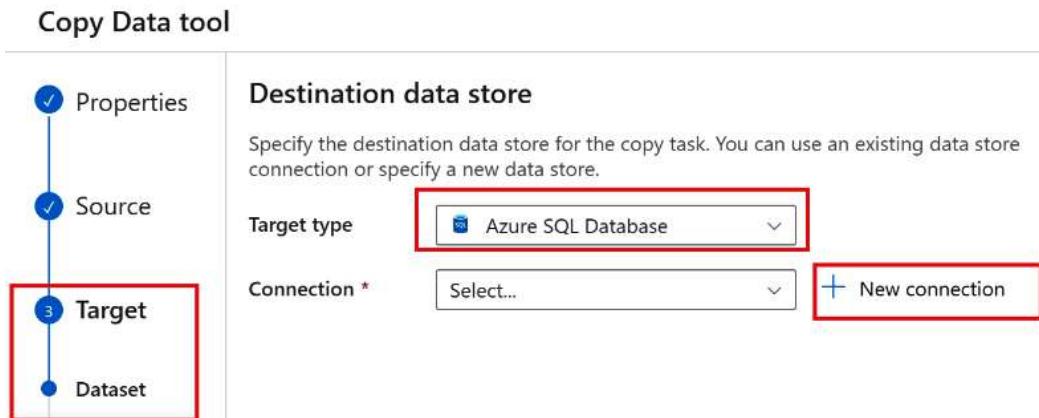


Figure 4.14 – Azure SQL database

- VI. Provide the database details and the credentials (sqladmin / SQLPwdW5 !k), as shown in the following screenshot:

New connection (Azure SQL Database)

Name *
SQLConnection

Description

Connect via integration runtime * ⓘ
 selfhosted-onpremise selfhosted-azure managed instance managed instance - external endpoint
⚠ The credentials are stored in the machines of self-hosted integration runtime if you don't choose to store them in Azure Key Vault.

Connection string **Azure Key Vault**

Account selection method ⓘ
 From Azure subscription Enter manually

Azure subscription
Select all

Server name *
packadesql

Database name *
sample

Authentication type *
SQL authentication

User name *
sqladmin

Password **Azure Key Vault**

Password *
.....

✔ Connection successful

Create **Back** **Test connection** **Cancel**

Figure 4.15 – Azure SQL database credentials

VII. Provide a destination table name. Check the **Skip column mapping for all tables** checkbox:

The screenshot shows the 'Copy Data tool' interface with the following steps listed on the left: Properties, Source, Target, Dataset, Configuration, Settings, and Review and finish. The 'Target' step is currently active. In the main pane, under 'Destination data store', the 'Target type' is set to 'Azure SQL Database' and the 'Connection' is 'SQLConnection'. The 'Integration runtime' is 'selfhosted-onpremise'. The 'Source' is 'File system file' and the 'Target' is 'dbo.orderdtls' (auto-create). At the bottom, there is a checked checkbox labeled 'Skip column mapping for all tables'. Navigation buttons 'Previous' and 'Next >' are at the bottom.

Figure 4.16 – Table details

VIII. Click through the **Configuration** and **Review and finish** screens to complete building the pipeline. Notice that the pipeline was created successfully and moved the data as well:



Deployment complete

Deployment step	Status
Validating copy runtime environment	✓ Succeeded
Creating datasets	✓ Succeeded
Creating pipelines	✓ Succeeded
Running pipelines	✓ Succeeded

Datasets and pipelines have been created. You can now monitor and edit the copy pipelines or click finish to close Copy Data Tool.

Figure 4.17 – Data transfer completion

Thus, we have successfully copied a file from an on-premises file system to Azure SQL Database using a self-hosted IR.

How it works...

Self-hosted IRs act as the gateway between Azure and the on-premise data source. Installing the **Microsoft IR** on the **SelfHostedIR1** machine ensured that the libraries and components required for moving the data from the on-premises file system to Azure are now available on **SelfHostedIR1**. Providing the self-hosted IR key during the installation registered the **SelfHostedIR1** machine in Azure as an IR. Once the self-hosted IR was configured, we were successfully able to move the file stored in **SelfHostedIR1** to Azure SQL Database (`packadesql.database.windows.net`) – Sample database.

The health of the IR machine plays an important role in the performance and stability of the data movement pipelines. It is recommended to monitor the self-hosted IR machine closely and upsize its configuration if required.

Configuring a shared self-hosted IR

A shared self-hosted IR, as the name suggests, can be shared among more than one data factory. This helps us use a single self-hosted IR to run multiple pipelines. In this activity, we'll learn how to share a self-hosted IR.

Getting ready

To get started, do the following:

1. Log in to `https://portal.azure.com` using your Azure credentials.
2. Open a new PowerShell prompt in your machine. Execute the following command to log in to your Azure account from PowerShell:

Connect-AzAccount

3. You will need an existing Data Factory account. If you don't have one, create one by executing the following PowerShell script:

```
$resourceGroupName = "PacktADE";
$location = 'east us'
$dataFactoryName = "ADFPacktADE2";
$DataFactory = Set-AzDataFactoryV2 -ResourceGroupName
$resourceGroupName -Location $location -Name
$dataFactoryName
```

4. You need a self-hosted IR. If you don't have one, follow the *Configuring a self-hosted IR* recipe in this chapter to create one.

How to do it...

Perform the following steps to create a linked self-hosted IR:

1. Let's create a new data factory account called ADFPacktSharedIR. The ADFPacktSharedIR data factory will use the IR from ADFPacktADE2. Use the following PowerShell command to create the ADFPacktSharedIR data factory:

```
$resourceGroupName = "PacktADE";
.setLocation = 'east us'
$dataFactoryName = "ADFPacktSharedIR";
$DataFactory = Set-AzDataFactoryV2 -ResourceGroupName
$resourceGroupName -Location $location -Name
$dataFactoryName
```

2. In the Azure portal, open ADFPacktADE2, and then open the **Manage** tab. On the **Manage** tab, select **Integration runtimes**.
3. Select the **selfhosted-onpremise** IR:

The screenshot shows the Microsoft Azure portal interface for the ADFPacktADE2 Data Factory. The left sidebar has a red box around the 'Source control' icon. The main area shows the 'Manage' tab selected. Under 'Integration runtimes', there is a table with two rows:

Name	Type
AutoResolveIntegrationRuntime	Azure
selfhosted-onpremise	Self-Hosted

Figure 4.18 – Opening the packtdatafactory Manage tab

Note

This recipe requires an existing self-hosted IR. If you don't have one, follow the previous recipe to create one.

4. In the **Edit integration runtime** window, select the **Sharing** tab. Click **Grant permission to another Data Factory or user-assigned managed identity**:

Edit integration runtime

Settings Nodes Auto update **Sharing** Links

You can share your self-hosted integration runtime (IR) with another Data Factory.

To enable sharing:

1. Grant permission to the Data Factory in which you would like to reference
2. Copy the below 'Resource ID' and use it while creating a new linked self-hosted Data Factory.

Resource ID ⓘ

/subscriptions/ /resourcegroups/PacktAD

+ Grant permission to another Data Factory or user-assigned managed identity

Resource name

Resource type

Apply

Cancel

Figure 4.19 – Granting permission to another data factory

5. In the **Assign permissions** window, type ADFPacktSharedIR in the search box, and then select ADFPacktSharedIR from the list. Hit the **Apply** button:

Edit integration runtime

Settings Nodes Auto update **Sharing** Links ⓘ

You can share your self-hosted integration runtime (IR) with another Data Factory.

To enable sharing:

1. Grant permission to the Data Factory in which you would like to reference this IR (shared).
2. Copy the below 'Resource ID' and use it while creating a new linked self-hosted IR in the other Data Factory.

Resource ID ⓘ

/subscriptions/ <i>I</i>	/resourcegroups/PacktADE/providers/	
--------------------------	-------------------------------------	--

Grant permission to another Data Factory or user-assigned managed identity Refresh

Resource name	Resource type	Remove
ADFPacktSharedIR	Data Factory	

Apply Cancel

Figure 4.20 – The IR setup window

- Note the resource ID for use in later steps. Click **Apply** to save the configuration settings.
6. In the Azure portal, open the **Manage** tab of **ADFPacktSharedIR**. On the **Connections** tab, select **Integration runtimes**. In the **Integration runtimes** panel, click **+ New**:

The screenshot shows the Microsoft Azure Data Factory interface. On the left, there's a sidebar with icons for Home, Data Factory, Connections, Linked services, Integration runtimes (which is highlighted with a red box), Azure Purview, Source control, Git configuration, ARM template, and Author. The main area is titled 'Integration runtimes' and contains the following text: 'The integration runtime (IR) is the compute'. Below this are two buttons: '+ New' (also highlighted with a red box) and 'Refresh'. There's also a 'Filter by name' input field. A list of items is shown with one item: 'AutoResolveIntegrationRuntime'. The entire screenshot is framed by a red border.

Figure 4.21 – Adding the IR to packtdatafactory2

7. In the **Integration runtime setup** window, select **Azure, Self-Hosted**:

Integration runtime setup

Integration Runtime is the native compute used to execute or dispatch activities. Choose what integration runtime to create based on required capabilities. [Learn more](#)

The screenshot shows the 'Integration runtime setup' window with two options listed:

- Azure, Self-Hosted**: This option is highlighted with a red box. It includes a blue cloud icon with a green plus sign and the text 'Perform data flows, data movement and dispatch activities to external compute.'
- Azure-SSIS**: This option includes a blue cylinder icon with a green plus sign and the text 'Lift-and-shift existing SSIS packages to execute in Azure.'

Figure 4.22 – Selecting the IR type

Click **Continue** to go to the next step.

8. In the next window, under **External Resources**, select **Linked Self-Hosted**:

Integration runtime setup

Network environment:

Choose the network environment of the data source / destination or external compute to which the integration runtime will connect to for data flows, data movement or dispatch activities:

Azure
 Use this for running data flows, data movement, external and pipeline activities in a fully managed, serverless compute in Azure.

Self-Hosted
 Use this for running activities in an on-premises / private network
[View more](#)

External Resources:

You can use an existing self-hosted integration runtime that exists in another resource. This way you can reuse your existing infrastructure where self-hosted integration runtime is setup.



Linked Self-Hosted
[Learn more](#)

Continue

Back

Cancel

Figure 4.23 – Selecting Linked Self-Hosted

Click **Continue** to go to the next step.

- In the next window, name the IR **selfhosted-onpremise**. Under **Resource ID**, enter the **Resource ID** value from step 5:

Integration runtime setup

Use an existing self-hosted integration runtime infrastructure in another Data Factory. ⓘ
This will create a logical link to an existing self-hosted integration runtime.

Name * ⓘ

selfhosted-onpremise

Description

Enter description here...

Type

Self-Hosted (Linked)

Resource ID * ⓘ

/subscriptions/
/resourcegroups/PacktADE/providers/Microsoft.DataFactory/factories/ADFPacktADE2/integrationruntimes/selfhosted-onpremise

Authentication method

- System-assigned Managed Identity (ADFPacktSharedIR)
- User Assigned Managed Identity (Preview)

Create

Back

Cancel

Figure 4.24 – Providing a shared IR resource ID

The IR is added to ADFPacktSharedIR. The **selfhosted-onpremise** IR is now shared between ADFPacktADE2 and ADFPacktSharedIR. The pipelines from the two data factories can benefit from one self-hosted IR.

Configuring high availability for a self-hosted IR

A self-hosted IR is a critical component for transferring data from on-premises data sources or data sources in virtual networks to Azure. Configuring additional nodes for a self-hosted IR ensures there is no single point of failure and provides high availability. Having additional nodes makes the self-hosted IR function as usual even when one of the nodes is under maintenance. Configuring additional nodes also allows load sharing with multiple data transfer jobs being executed in parallel on different nodes.

In the following recipe, we will install an IR on an additional machine to provide high availability and load-sharing capabilities.

Getting ready

Perform the following steps before working on this recipe:

1. Log in to `https://portal.azure.com` using your Azure credentials.
2. You will need an existing Data Factory account. If you don't have one, create one by executing the following PowerShell script. Open a new PowerShell prompt. Execute the following commands to log in to your Azure account from PowerShell and create a Data Factory account:

```
Connect-AzAccount
$resourceGroupName = "PacktADE";
.setLocation = 'east us'
$dataFactoryName = "ADFPacktADE2";
$DataFactory = Set-AzDataFactoryV2 -ResourceGroupName
$resourceGroupName -Location $location -Name
$dataFactoryName
```

3. Configure a self-hosted IR, as explained in the *Configuring a self-hosted IR* section of this chapter.
4. You will need an additional machine to be used as a secondary node for the IR.

How to do it...

Perform the following steps to configure high availability for a self-hosted IR:

1. Log in to `portal.azure.com`. Go to **Data Factory**. Click **Open Azure Data Factory Studio**.
2. Click on **Manage -> Integration runtimes**. Make note of the version number of the self-hosted IR. Click on **selfhosted-onpremise**, as shown in the following screenshot:

The screenshot shows the 'Integration runtimes' page in the Azure portal. On the left, there's a sidebar with icons for Home, Connections, Author, Linked services, Integration runtimes (which is selected and highlighted with a red box), Microsoft Purview, Source control, Git configuration, ARM template, Author, and Triggers. The main area has a title 'Integration runtimes' and a subtitle: 'The integration runtime (IR) is the compute infrastructure to provide the following data integration capabilities across different network environments'. Below that are 'New' and 'Refresh' buttons, a 'Filter by name' input field, and a table header: 'Showing 1 - 2 of 2 items'. The table columns are: Name, Type, Sub-type, Status, Related, Region, Version. There are two rows: one for 'AutoResolve[IntegrationR...' (Azure, Public, Running, 0, Auto Resolve, ---, ---) and one for 'selfhosted-onpremise' (Self-Hosted, ---, Running, 2, ---, 5.19.8214.2). The 'selfhosted-onpremise' row is also highlighted with a red box.

Figure 4.25 – The self-hosted IR

3. Copy the IR key and save it in Notepad for future reference.

Edit integration runtime

Settings Nodes Auto update Sharing Links

Install integration runtime on Windows machine or add further nodes using the Authentication Key.

Name ⓘ

selfhosted-onpremise

Description

Option 1: Express setup

[Click here to launch the express setup for this computer](#)

Option 2: Manual setup

Step 1: [Download and install integration runtime](#)

Step 2: Use this key to register your integration runtime

Name	Authentication key
------	--------------------

Key1	IR@42b666c7-2d9e-4e28-9fe3-6383f169383c@ADFPacktADE2@ServiceEr		
Key2	IR@42b666c7-2d9e-4e28-9fe3-6383f169383c@ADFPacktADE2@ServiceEr		



Figure 4.26 – Copy the key

4. Click on **Download and install integration runtime** under **Option 2: Manual setup**.

Edit integration runtime

Settings Nodes Auto update Sharing Links

Install integration runtime on Windows machine or add further nodes using the Authentication Key.

Name ⓘ

selfhosted-onpremise

Description

Option 1: Express setup

[Click here to launch the express setup for this computer](#)

Option 2: Manual setup

Step 1: [Download and install integration runtime](#)

Step 2: Use this key to register your integration runtime

Name	Authentication key	Actions
Key1	IR@42b666c7-2d9e-4e28-9fe3-6383f169383c@ADFPacktADE2@ServiceEr	
Key2	IR@42b666c7-2d9e-4e28-9fe3-6383f169383c@ADFPacktADE2@ServiceEr	

Figure 4.27 – Download the IR

5. Download the version of the IR noted in *step 2*. In my case, it is **IntegrationRuntime_5.19.8214.2.msi**.

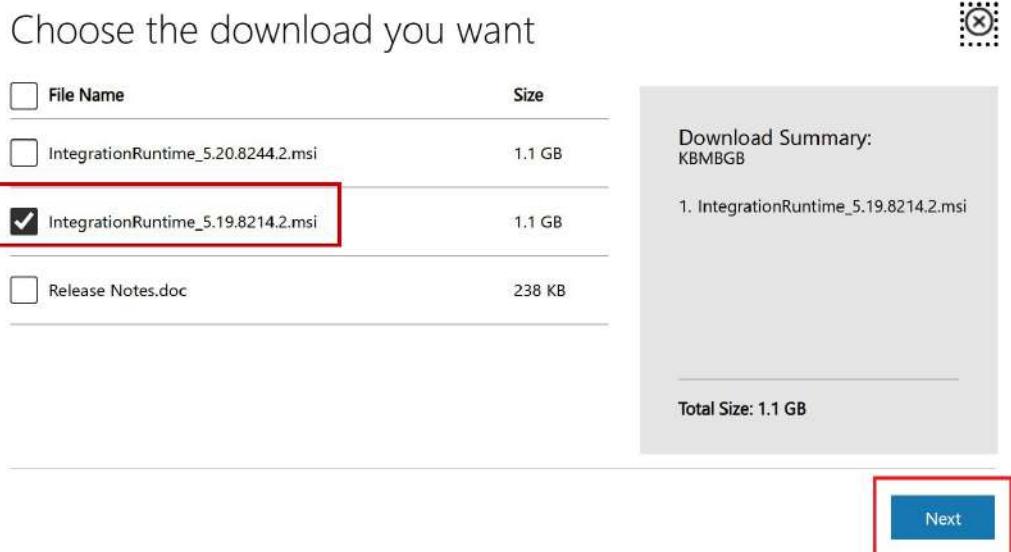


Figure 4.28 – Choose the IR version

- To add a secondary node, the **Remote access from intranet** setting needs to be enabled on the active node of the self-hosted IR machine via the Microsoft Integration Runtime tool. Log in to the **SelfhostedIR1** machine. Go to the **Start** menu and find **Microsoft Integration Runtime**. On the **Settings** tab, hit the **Change** button to enable **Remote access from intranet**:

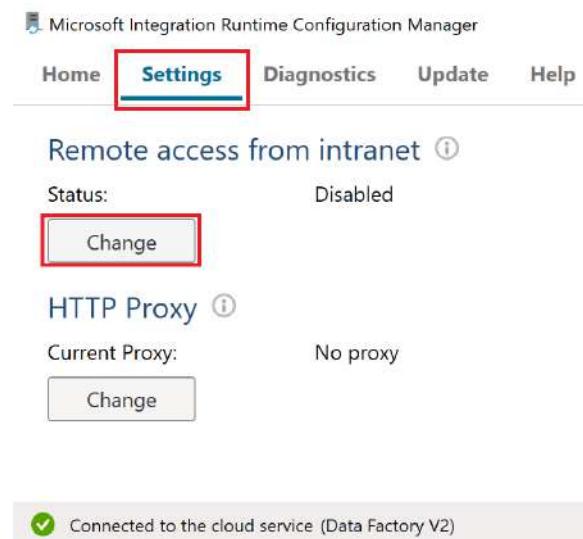


Figure 4.29 – Enabling remote access in the primary node

7. Select the **Enable without TLS/SSL certificate (Basic)** option. Make note of the TCP port 8060 used by the IR. Hit the **OK** button. The integration service on the primary node will restart for the change to take effect:

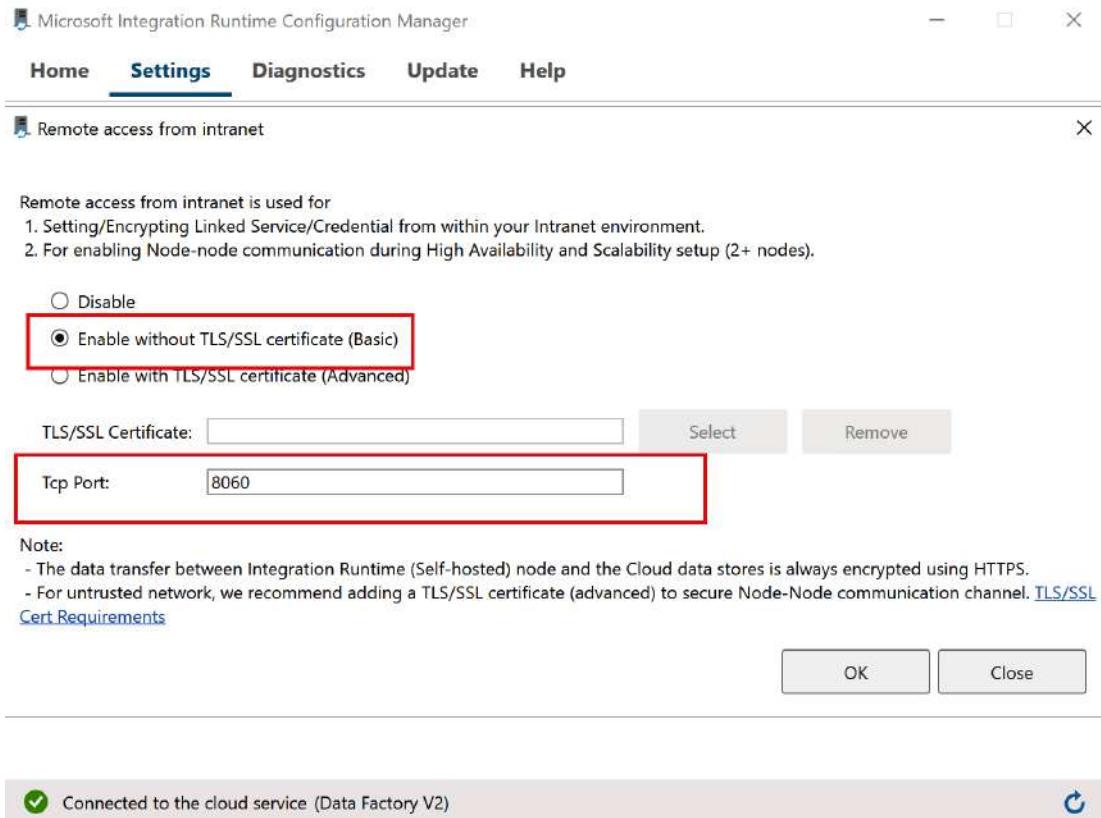


Figure 4.30 – Enabling the basic option

8. Log into the machine that'll be used as the additional node as secondary node for the IR. Copy the **IntegrationRuntime_5.19.8214.2.msi** file downloaded to the machine to be used as a secondary node for the IR. In this recipe, the machine name is **SelfHostedIR2**.
9. Install the **IntegrationRuntime_5.19.8214.2.msi** file in **SelfHostedIR2**. At the end of the installation, you will be prompted to register the key, which we copied in *step 3*. Paste the key and click on the **Register** button:

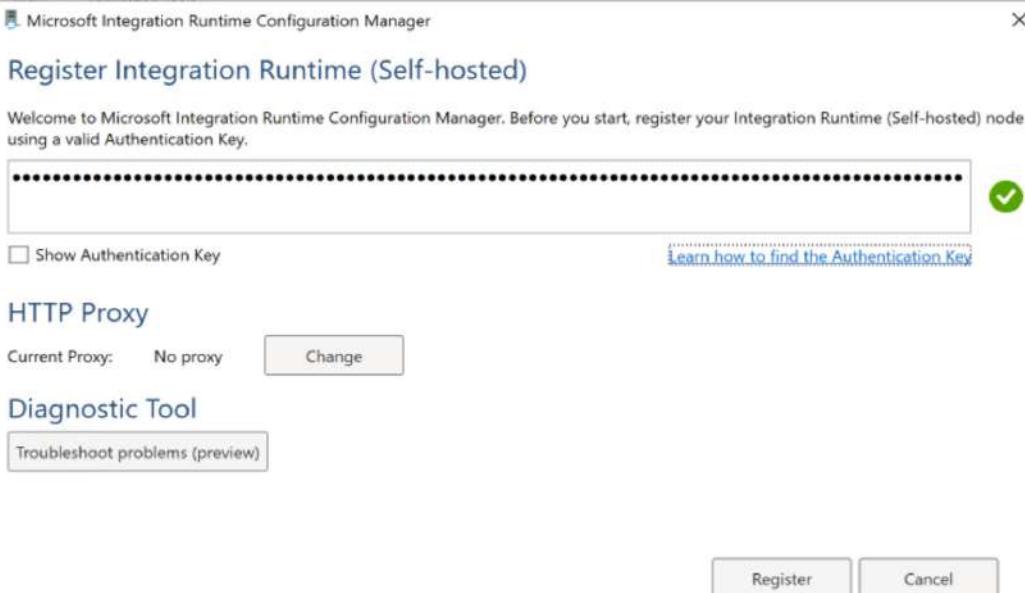


Figure 4.31 – Registering the key

10. Click on **Enable remote access from intranet** and click the **Next** button:

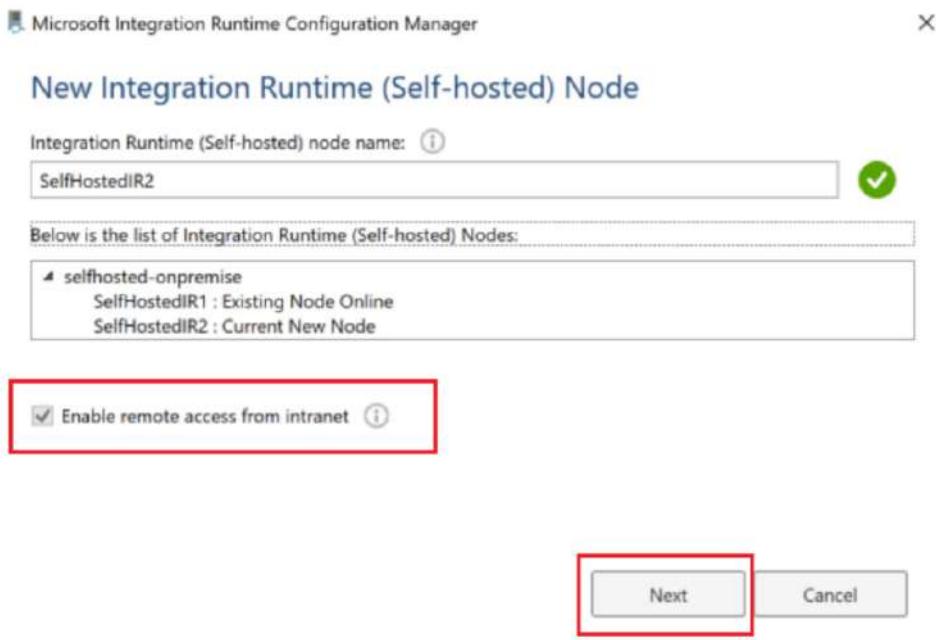


Figure 4.32 – Adding the secondary node

11. Hit the **Finish** button. Ensure to provide the same port number as the active node (the default is 8060). Ensure that port 8060 is open for communication between active and passive nodes and is not blocked by a firewall.

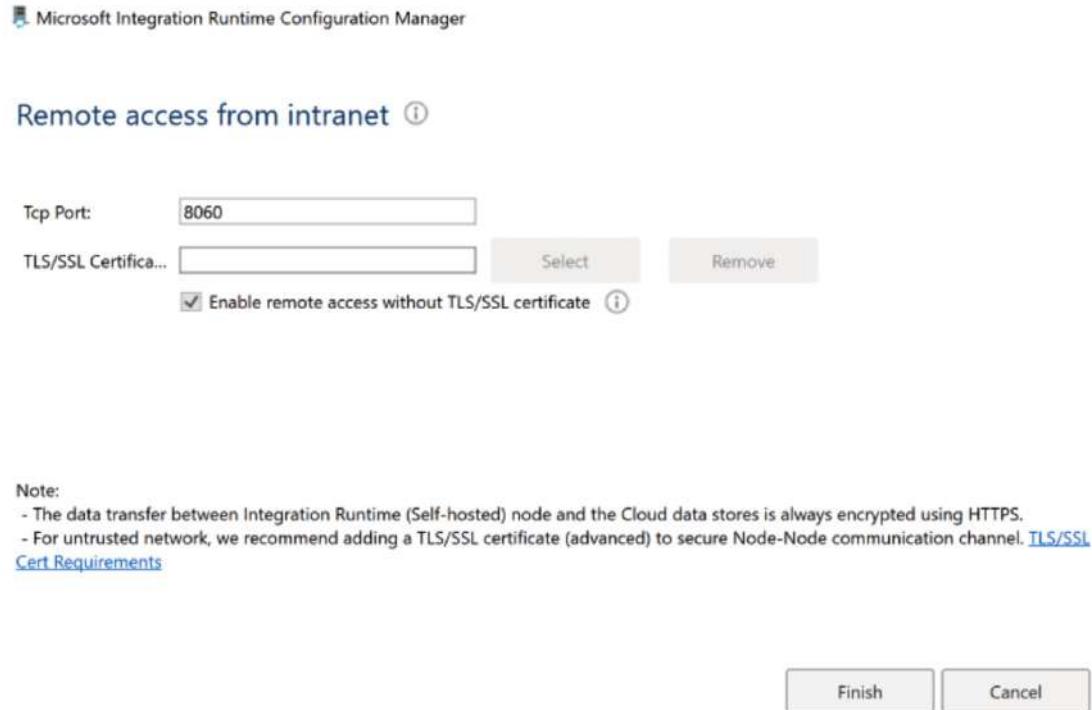


Figure 4.33 – Completing adding the secondary node

12. You should get the following screen after a successful registration. Hit the **Close** button to complete the process:

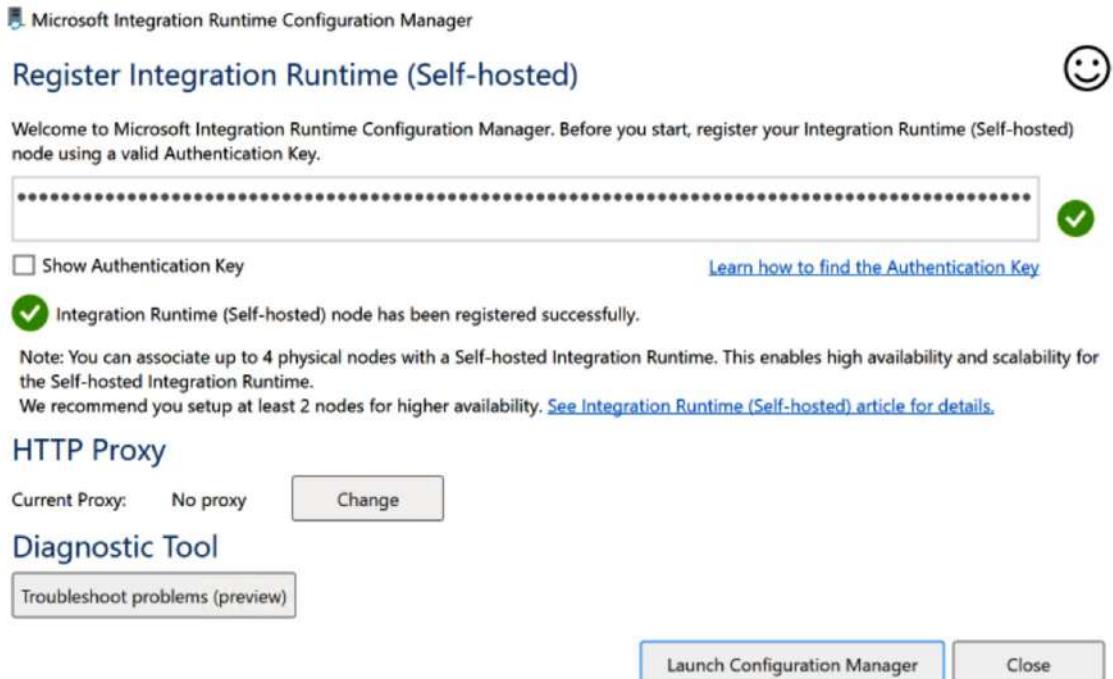


Figure 4.34 – Secondary node registration confirmation

13. You may verify the successful addition of a secondary node in `portal.azure.com`. Go to the Data Factory account created in this chapter. Click **Open Azure Data Factory Studio**. Click **Manage** -> **Integration runtimes** -> **selfhosted-onpremise**. Click the **Nodes** tab. Notice that the second node has been successfully added:

The screenshot shows the 'Edit integration runtime' blade in the Azure Data Factory studio. The 'Nodes' tab is selected. It lists two nodes: 'SelfHostedIR1' and 'SelfHostedIR2'. Both nodes are marked as 'Running'. The 'selfhosted-onpremise' node is highlighted with a red box. The left sidebar shows navigation options like Data Factory, Connections, Linked services, and Integration runtimes.

Name	Status	IP address
SelfHostedIR1	Running	Get IP address
SelfHostedIR2	Running	Get IP address

Figure 4.35 – Verifying the secondary node on the Azure portal

How it works...

Downloading the IR and installing it on a machine provides the libraries that are required for moving the data between an on-premises data source and Azure. Registering the key from the self-hosted IR that has already been provisioned in your Data Factory account helps the secondary node to identify the Data Factory account and the self-hosted IR, which it needs to register with and get provisioned as a secondary node to provide high availability and load-sharing capabilities. As of December 2021, you may add up to four nodes within a self-hosted IR. Having four nodes allows more data flows to be processed in parallel and offers greater high availability.

Patching a self-hosted IR

A self-hosted IR is a key component in Azure Data Factory, facilitating data movement between on-premises data sources and Azure. The self-hosted IR is configured by installing the Microsoft Integration Runtime installer on a machine that can connect to an on-premises data source and Azure. Keeping the self-hosted IR updated with the latest version is of the utmost importance to keep it bug-free and secure.

By default, a self-hosted IR has automatic updates enabled, which will apply the patches and keep the self-hosted IR updated. However, many users of Azure Data Factory would prefer to apply the patches themselves, as they would like to test their data transfer tasks and pipelines in a **User Acceptance Test (UAT)** environment with the latest patches, before running them in a production environment. So, in this recipe we will perform the following:

- Disabling the automatic patch update
- Installing the latest patches for a self-hosted IR

Getting ready

Perform the following steps before working on this recipe:

1. Log in to <https://portal.azure.com> using your Azure credentials.
2. You will need an existing Data Factory account. If you don't have one, create one by executing a PowerShell script. Open a new PowerShell prompt. Execute the following commands to log in to your Azure account from PowerShell and create a Data Factory account:

```
Connect-AzAccount  
$resourceGroupName = "PacktADE";  
$location = 'east us'  
$dataFactoryName = "ADFPacktADE2";
```

```
$DataFactory = Set-AzDataFactoryV2 -ResourceGroupName  
$resourceGroupName -Location $location -Name  
$dataFactoryName
```

3. Configure the self-hosted IR, as explained in the *Configuring a self-hosted IR* section of this chapter.

How to do it...

We have to perform two major tasks – disabling the auto update, followed by applying the latest patch. Let's perform the following steps to complete the tasks:

1. Log in to portal.azure.com. Go to Data Factory. Click **Open Azure Data Factory Studio**.
2. Click **Manage** -> **Integration runtimes**. Make a note of the version number of the self-hosted IR. Click on **selfhosted-onpremise**:

The screenshot shows the Azure Data Factory Studio interface. On the left, there is a sidebar with icons for Home, Connections, Author, Linked services, Integration runtimes (which is selected and highlighted with a red box), Microsoft Purview, Source control, Git configuration, ARM template, Author, and Triggers. The main area is titled "Integration runtimes" and contains the following text: "The integration runtime (IR) is the compute infrastructure to provide the following data integration capabilities across different network environment". Below this is a "+ New" button and a "Refresh" button. A search bar labeled "Filter by name" is present. A table titled "Showing 1 - 2 of 2 items" lists two entries:

Name	Type	Sub-type	Status	Related	Region	Version
AutoResolveIntegrationR...	Azure	Public	Running	0	Auto Resolve	---
selfhosted-onpremise	Self-Hosted	---	Running	2	---	5.19.8214.2

Figure 4.36 – Patching a self-hosted IR

3. Click **Auto update**. Set **Auto update** to **Disable** and hit the **Apply** button:

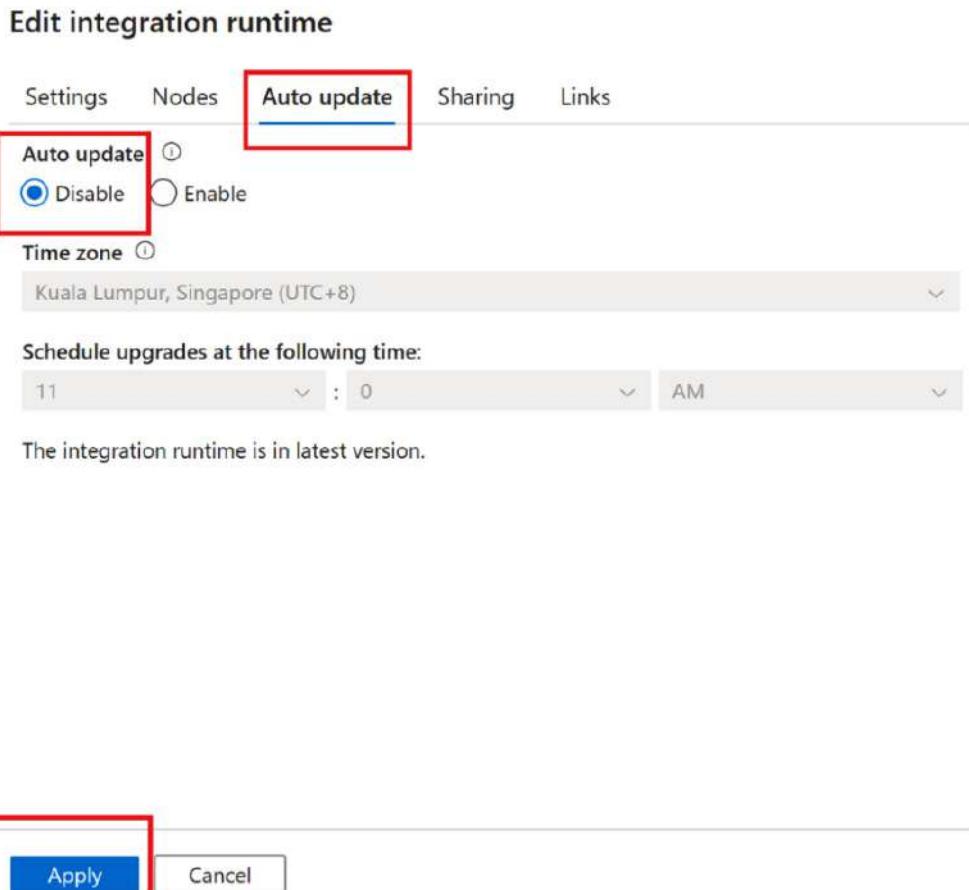


Figure 4.37– Disabling an auto update

4. To update the IR, click **selfhosted-onpremise** in **Integration runtimes** again.
5. Click **Download and install integration runtime** under **Option 2: Manual setup:**

Edit integration runtime

Settings Nodes Auto update Sharing Links

Install integration runtime on Windows machine or add further nodes using the Authentication Key.

Name ⓘ

selfhosted-onpremise

Description

Option 1: Express setup

[Click here to launch the express setup for this computer](#)

Option 2: Manual setup

Step 1: [Download and install integration runtime](#)

Step 2: Use this key to register your integration runtime

Name	Authentication key	Actions
Key1	IR@42b666c7-2d9e-4e28-9fe3-6383f169383c@ADFPacktADE2@ServiceEr	
Key2	IR@42b666c7-2d9e-4e28-9fe3-6383f169383c@ADFPacktADE2@ServiceEr	

Name	Authentication key	Actions
Key1	IR@42b666c7-2d9e-4e28-9fe3-6383f169383c@ADFPacktADE2@ServiceEr	
Key2	IR@42b666c7-2d9e-4e28-9fe3-6383f169383c@ADFPacktADE2@ServiceEr	

Figure 4.38 – Downloading the patches

6. Select the latest version to download, as long as it is higher than the version of self-hosted IR you recorded in *step 2*:

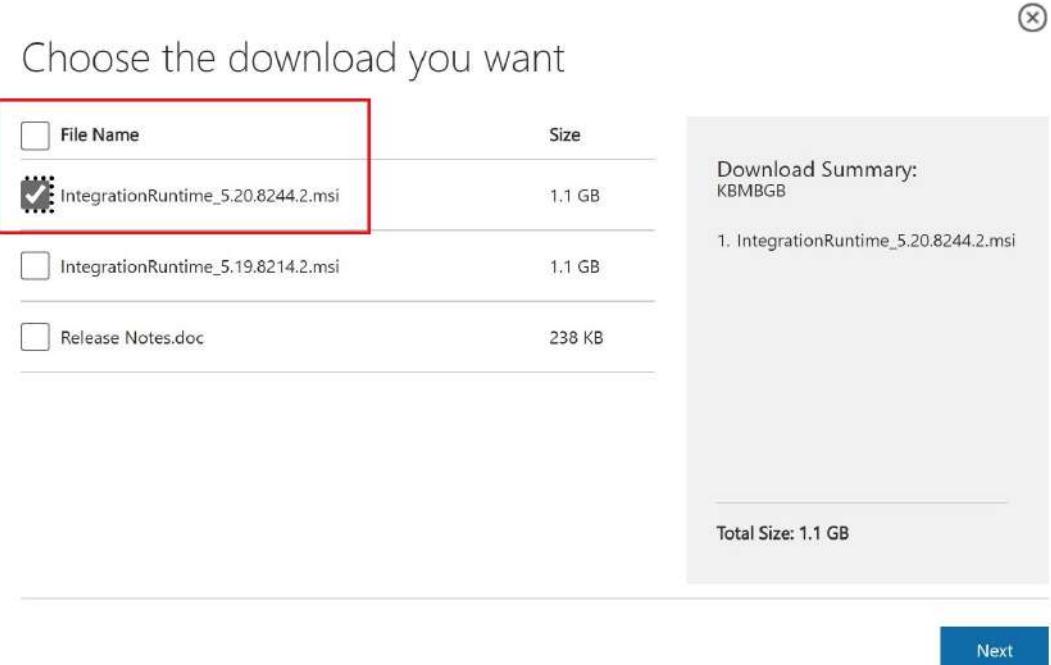


Figure 4.39 – Downloading the latest patches

7. Copy the installer to the self-hosted runtime machine and follow the installation instructions on the screen to complete the installation of the .msi file:

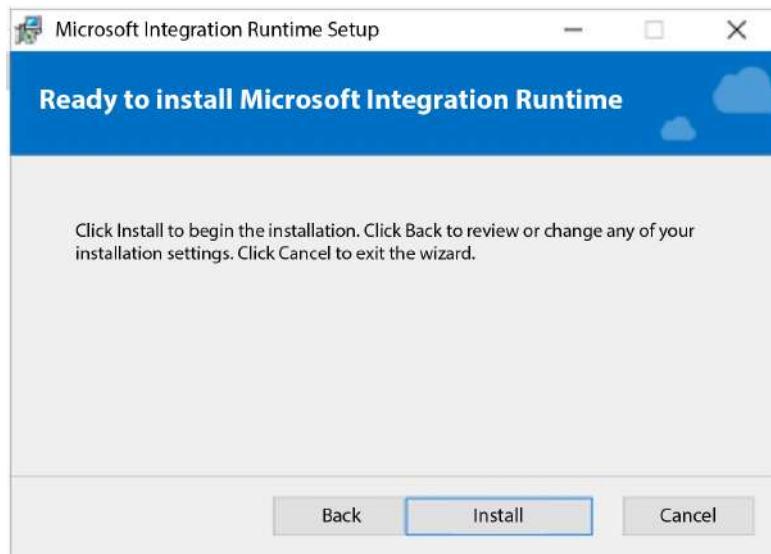


Figure 4.40 – Installing the MSI package

8. Upon completion of the installation, the **Microsoft Integration Runtime** application will automatically open and show that it is successfully connected to Azure Data Factory, as shown in the following screenshot:

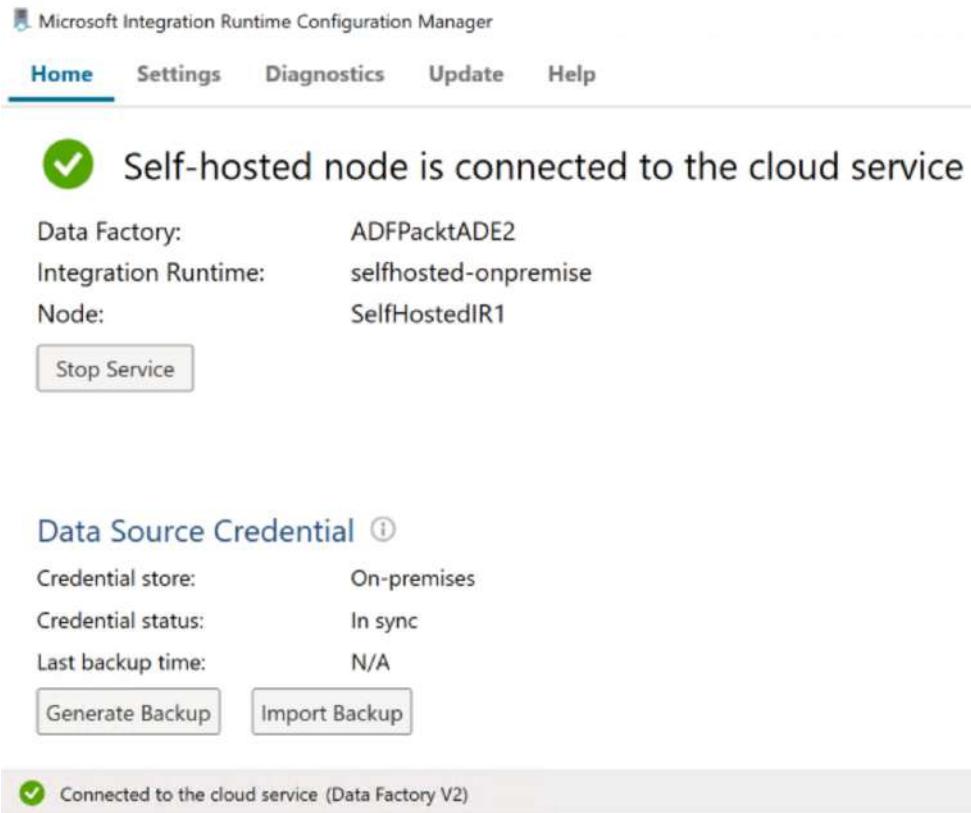


Figure 4.41 – Patch installation completion

9. To confirm the patch has been applied successfully, let's verify it from Azure portal. Go to portal.azure.com. Go to the Data Factory account that you created earlier in this chapter. Click **Open Azure Data Factory Studio**. Click the **Monitor** button on the left-hand side of the screen. Click **Integration runtimes** and then click **selfhosted-onpremise**:

Name	Type
selfhosted-onpremise	Azure

Figure 4.42 – Verifying the patch installation

In the **Node Details** section, verify whether the version number of the node has been updated to the version that was installed:

Node Details					
Name	Status	Version	Available memory	CPU utilization	Network (In/Out)
SelfHostedIR1	Running	5.20.8244.2	4528MB	64%	2.05KBps/6.47KBps

Figure 4.43 – Verifying patch installation with the version number

10. Repeat the preceding nine steps for each node in your self-hosted IR if your self-hosted integration setup contains multiple nodes.

How it works...

Disabling the auto update of the self-hosted IR ensures that patches are not rolled out automatically. While auto update is convenient, applying patches manually gives better control, especially when we are running mission-critical pipelines, where testing in a UAT environment is mandatory before the production rollout.

Downloading the latest patches and applying them on **SelfHostedIR1** works smoothly and doesn't demand providing the IR keys again, as the self-hosted IR is already connected to Azure Data Factory. Post-update, the version number of the node installed is also reflected in Data Factory Studio in the Azure portal in the **Monitoring** section for the self-hosted IR.

During the upgrade, active data flows may experience connectivity failures, and you are recommended to configure high availability, as explained in the *Configuring high availability for a self-hosted IR* section of this chapter.

Migrating an SSIS package to Azure Data Factory

SQL SSIS is a widely used on-premises **Extract, Transform, and Load** (ETL) tool. In this recipe, we'll learn how to migrate an existing SSIS package to Azure Data Factory.

We'll do this by configuring an Azure SSIS IR, uploading the SSIS package to the Azure SQL Database SSISDB, and then executing the package using the **Execute SSIS Package** activity.

Getting ready

To get started, do the following:

1. Open a new PowerShell prompt. Execute the following command to log in to your Azure account from PowerShell:

```
Connect-AzAccount
```

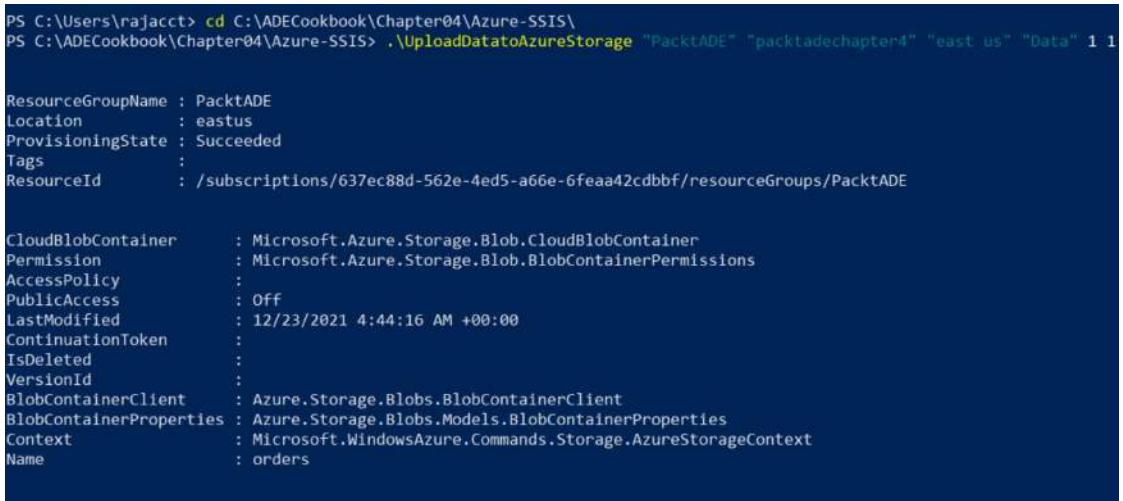
2. You will need an existing Data Factory account. If you don't have one, create one by executing the following PowerShell script:

```
$resourceGroupName = "PacktADE";
$location = 'east us'
$dataFactoryName = "ADFPacktADE2";
$dataFactory = Set-AzDataFactoryV2 -ResourceGroupName
$resourceGroupName -Location $location -Name
$dataFactoryName
```

3. Download the Azure-SSIS folder from <https://github.com/PacktPublishing/Azure-Data-Engineering-Cookbook-2nd-edition/tree/main/Chapter04/>. You may fork the entire repository to your GitHub account and download it as a ZIP file to your local machine.
4. In PowerShell, go to the Azure-SSIS folder and run the following command:

```
. \UploadDatatoAzureStorage "PacktADE" "packtadechapter4"  
"east us" "Data" 1 1
```

This command creates a storage account named packtadechapter4 and pushes all the files in the Data folder into the blob storage account. You should have an output similar to the one shown in the following screenshot:



```
PS C:\Users\rajacct> cd C:\ADECookbook\Chapter04\Azure-SSIS\  
PS C:\ADECookbook\Chapter04\Azure-SSIS> .\UploadDatatoAzureStorage "PacktADE" "packtadechapter4" "east us" "Data" 1 1  
  
ResourceGroupName : PacktADE  
Location         : eastus  
ProvisioningState : Succeeded  
Tags             :  
ResourceId       : /subscriptions/637ec88d-562e-4ed5-a66e-6feaa42cdbbf/resourceGroups/PacktADE  
  
CloudBlobContainer      : Microsoft.Azure.Storage.Blob.CloudBlobContainer  
Permission          : Microsoft.Azure.Storage.Blob.BlobContainerPermissions  
AccessPolicy        :  
PublicAccess        : Off  
LastModified       : 12/23/2021 4:44:16 AM +00:00  
ContinuationToken   :  
IsDeleted          :  
VersionId          :  
BlobContainerClient : Azure.Storage.Blobs.BlobContainerClient  
BlobContainerProperties : Azure.Storage.Blobs.Models.BlobContainerProperties  
Context            : Microsoft.WindowsAzure.Commands.Storage.AzureStorageContext  
Name               : orders
```

Figure 4.44 – A file upload to the blob storage

How to do it...

We'll start by creating a new Azure SSIS IR:

1. In the Azure portal, open the Data Factory **Manage** tab. Select **Integration runtimes**, and then select **+ New**. Provide the name, location, node size, and node number. We have kept the node size and node number to the smallest available ones to save costs:

Integration runtime setup

General settings

Name * ⓘ

Description ⓘ

Type

Location * ⓘ

Node size * ⓘ

Node number * ⓘ

Edition/license * ⓘ

Save money

Save with a license you already own. Already have a SQL Server license?

By selecting "yes", I confirm I have a SQL Server license with Software Assurance to apply this [Azure Hybrid Benefit for SQL Server](#).

Please be aware that the cost estimate for running your Azure-SSIS Integration Runtime is **(1 * US\$ 1.938)/hour = US\$ 1.938/hour**, see [here](#) for current prices.

Figure 4.45 – Configuring the Azure SSIS IR

Click **Continue** to go to the next step.

We can either host the SSIS package in an SSISDB database, hosted either on Azure SQL Database or SQL Managed Instance, or we can host the package on Azure Storage. In this recipe, we'll host the package on SSISDB.

2. In the **Deployment settings** section, check the **Create SSIS catalog...** checkbox and provide the Azure SQL Database details, as shown in the following screenshot. You may have to provision Azure SQL Database if you don't have these details. The PowerShell script to provision Azure SQL Database is provided at <https://github.com/PacktPublishing/Azure-Data-Engineering-Cookbook-2nd-edition/upload/main/Chapter04/Azure-SSIS/AzureSQLDB.ps1>. We have set **Catalog database service tier** to **Basic** to save on costs.

Integration runtime setup

Deployment settings

Create SSIS catalog (SSISDB) hosted by Azure SQL Database server/Managed Instance to store your projects/packages/environments/execution logs
(See more info [here](#))

Subscription * ⓘ

Visual Studio Ultimate with MSDN ()

Location ⓘ

East US

Catalog database server endpoint * ⓘ

packadesql.database.windows.net

Use AAD authentication with the system managed identity for Data Factory ⓘ
(See how to enable it [here](#))

Use AAD authentication with a user-assigned managed identity for Data Factory ⓘ
(See how to enable it [here](#))

Admin username * ⓘ

sqladmin

Admin password * ⓘ

.....

Use dual standby Azure-SSIS Integration Runtime pair with SSISDB failover ⓘ
(See more info [here](#))

Catalog database service tier * ⓘ

Basic

Create package stores to manage your packages that are deployed into file system/Azure Files/SQL Server database (MSDB) hosted by Azure SQL Managed Instance ⓘ
(See more info [here](#))

Continue

Back

Cancel

Figure 4.46 – Configuring the SSISDB catalog

Click **Continue** to go to the next step.

3. On the **Advanced settings** page, set parallel executions to **1** and uncheck the **Select a VNet for your Azure-SSIS Integration Runtime to join...** option, as shown in the following screenshot, and click **Continue** to create the Azure SSIS IR:

Integration runtime setup

Advanced settings

Maximum parallel executions per node * ⓘ

▼

Customize your Azure-SSIS Integration Runtime with additional system configurations/component installations ⓘ
(See more info [here](#))

Select a VNet for your Azure-SSIS Integration Runtime to join, allow ADF to create certain network resources, and optionally bring your own static public IP addresses ⓘ
(See more info [here](#))

Set up Self-Hosted Integration Runtime as a proxy for your Azure-SSIS Integration Runtime ⓘ
(See more info [here](#))

⚠ If access to your Azure SQL Database server is disabled from other Azure services/resources, please select a VNet for your Azure-SSIS Integration Runtime to join, so it can access SSISDB – Alternatively, please configure the "Firewall and virtual networks" settings of your Azure SQL Database server on Azure portal to enable the "Allow Azure services and resources to access this server" property.

Continue **Back** **Cancel**

Figure 4.47 – Configuring advanced settings

When created, the **AzureSSISIR** IR is now listed, as in the following screenshot:

Name	Type	Sub-type	Status
AutoResolveIntegrationRuntime	Azure	Public	Running
AzureSSISIR	Azure-SSIS	---	Running
selfhosted-onpremise	Self-Hosted	Shared	Running

Figure 4.48 – Viewing the Azure SSIS IR

- The next step is to deploy the SSIS package to the Azure SQL Database SSISDB catalog. To do that, open **SQL Server Management Studio (SSMS)**, and connect to the **SSISDB** in Azure SQL Database (`packadesql.database.windows.net`) in **Object Explorer**. In the **Object Explorer** window, expand the **Integration Services Catalogs** folder, right-click **SSISDB**, and then select **Create Folder...**:

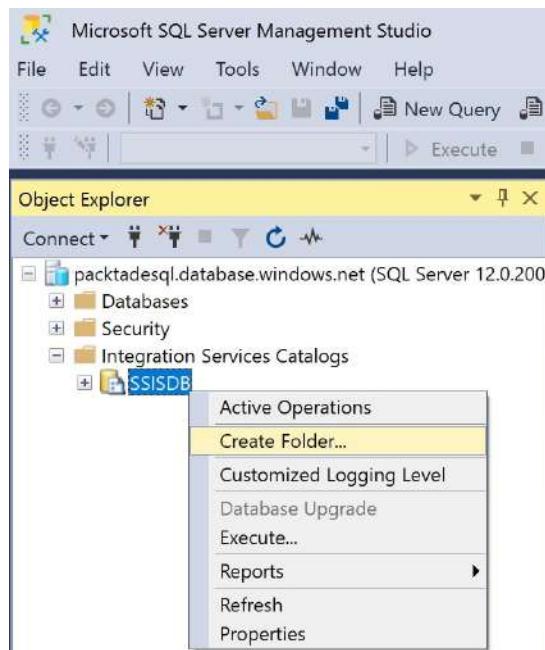


Figure 4.49 – Creating a folder in the SSISDB catalog

5. In the **Create Folder** dialog box, set **Folder name** to AzureSSIS:

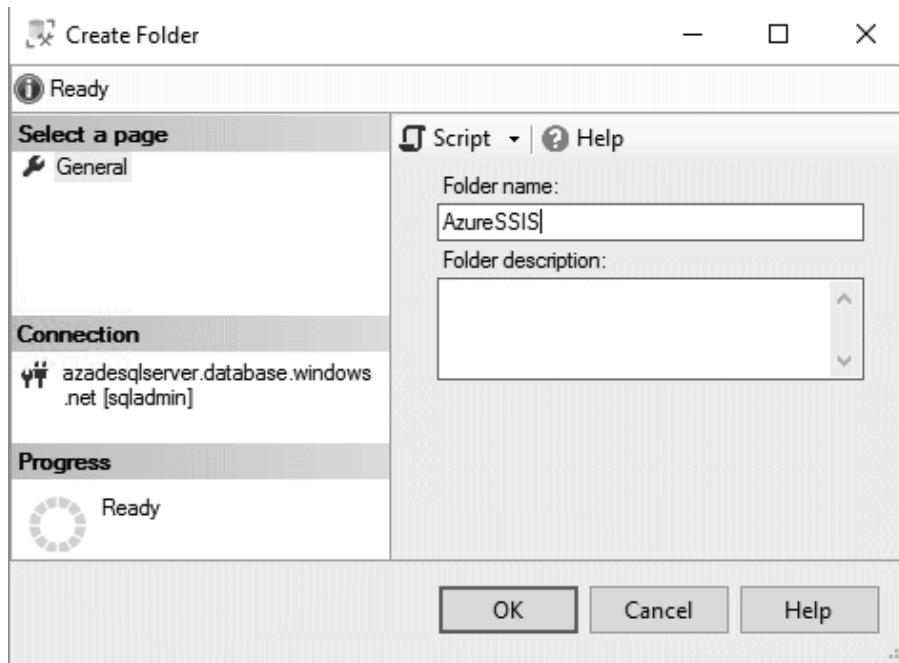


Figure 4.50 – Providing the folder settings

Click **OK** to create the folder.

6. In the **Object Explorer** window, expand the AzureSSIS folder, right-click **Project**, and select **Deploy Project...** from the context menu:

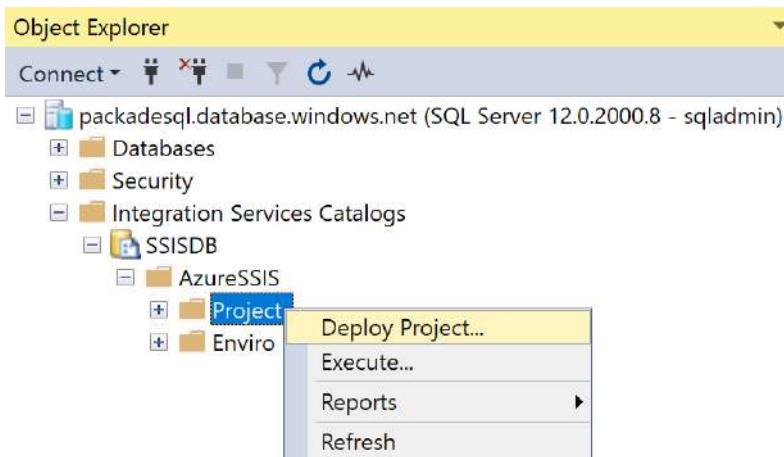


Figure 4.51 – Opening the SSIS deployment wizard

7. In the **Integration Services Deployment Wizard** window, select the **Select Source** tab, browse to the `\chapter4\Azure-SSIS\` path, and select the `CopyFiles.ispac` file. If you haven't downloaded the `CopyFiles.ispac` file, download it from <https://github.com/PacktPublishing/Azure-Data-Engineering-Cookbook-2nd-edition/tree/main/Chapter04/Azure-SSIS/>:

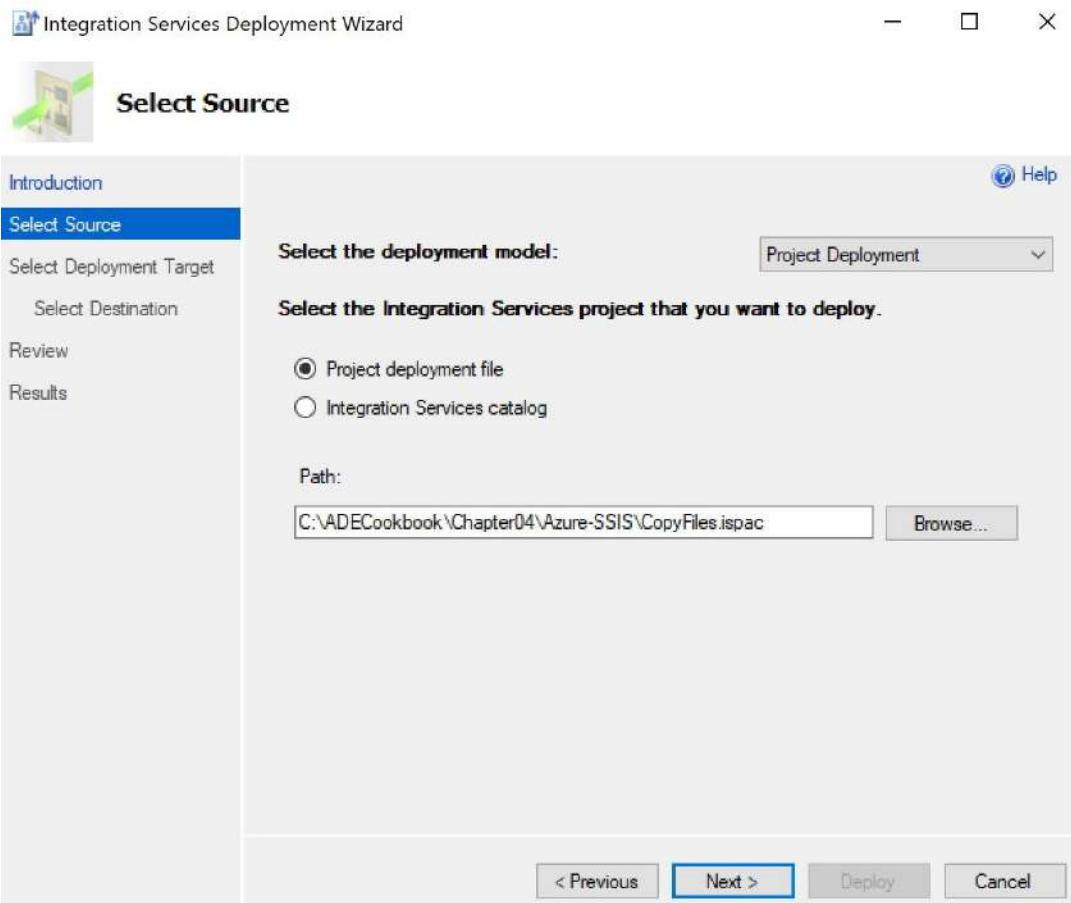


Figure 4.52 – Select Source in the Integration Services Deployment Wizard

Note

The SSIS package copies the files from an Azure storage account from the `orders/datain` folder to the `orders/dataout` folder. The Azure storage account name and key are passed as parameters.

Click **Next** to go to the next step.

8. In the **Select Deployment Target** tab, select **SSIS in Azure Data Factory**:

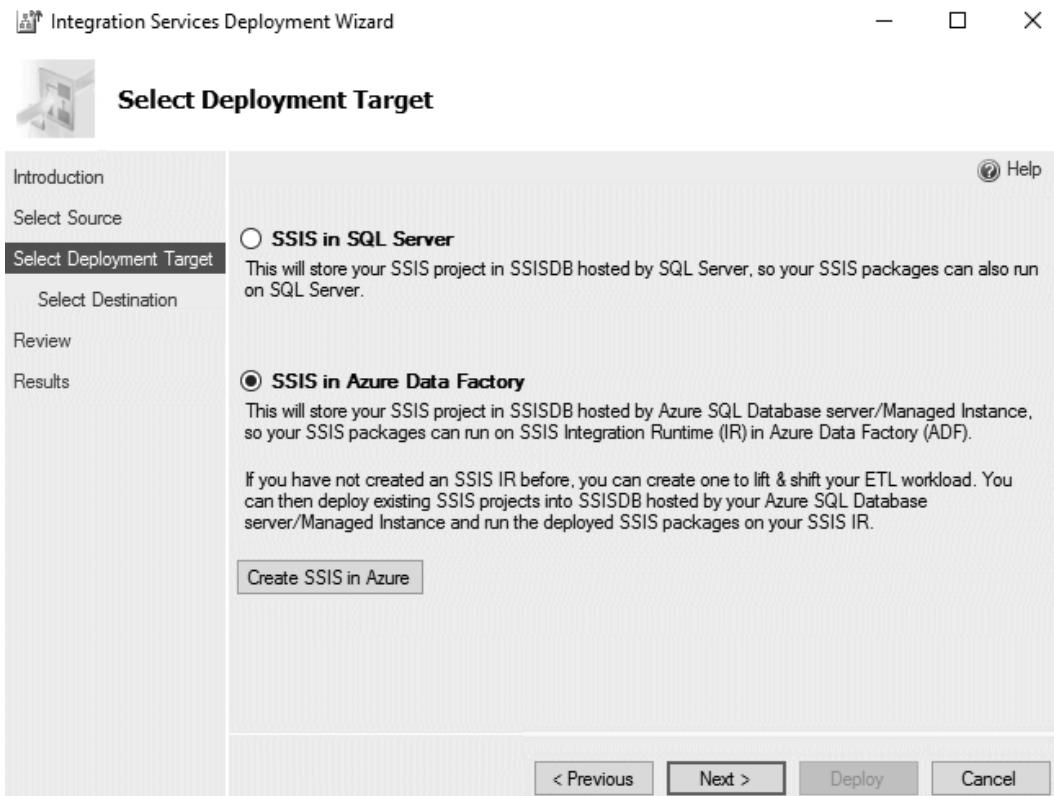


Figure 4.53 – Select Deployment Target in the Integration Services Deployment Wizard

Click **Next** to go to the next step.

9. In the **Select Destination** tab, provide the Azure SQL Server admin user and password, and click **Connect** to test the connection:

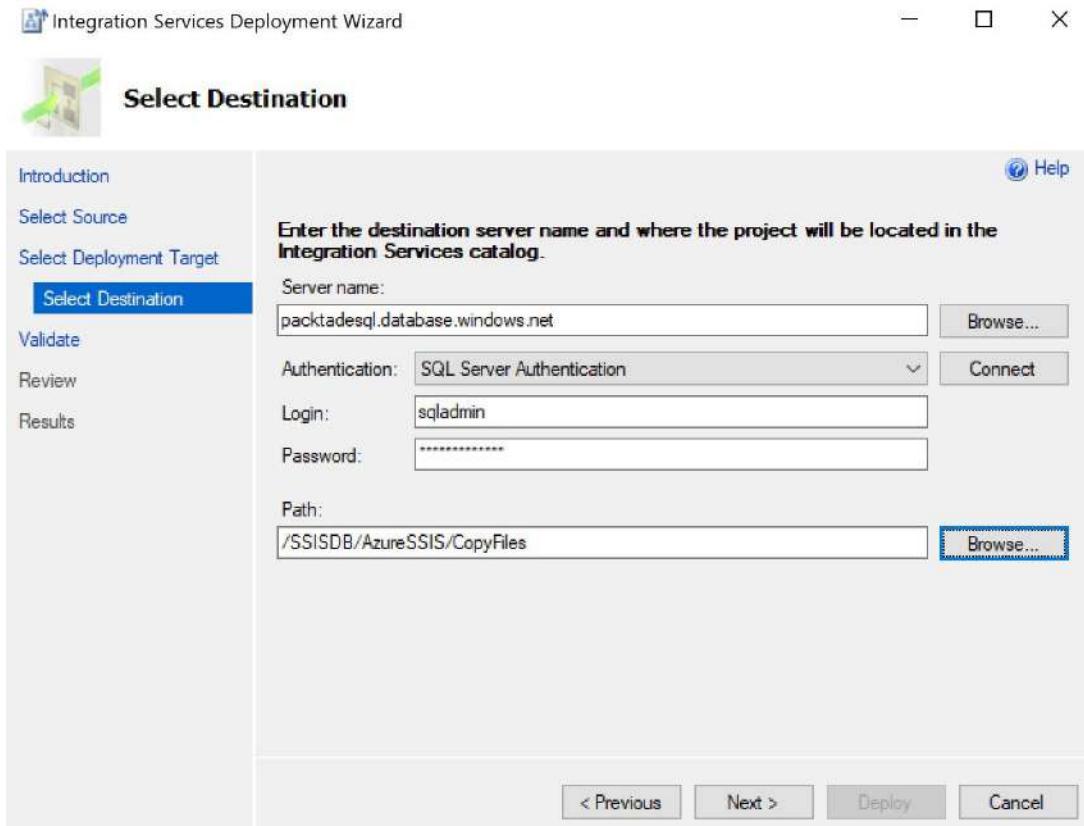


Figure 4.54 – Select Destination in the Integration Services Deployment Wizard

10. After a successful connection, click **Next**. The deployment wizard will validate that the package is without issues. After successful validation, click **Next** and then **Deploy** to complete the SSIS package deployment:

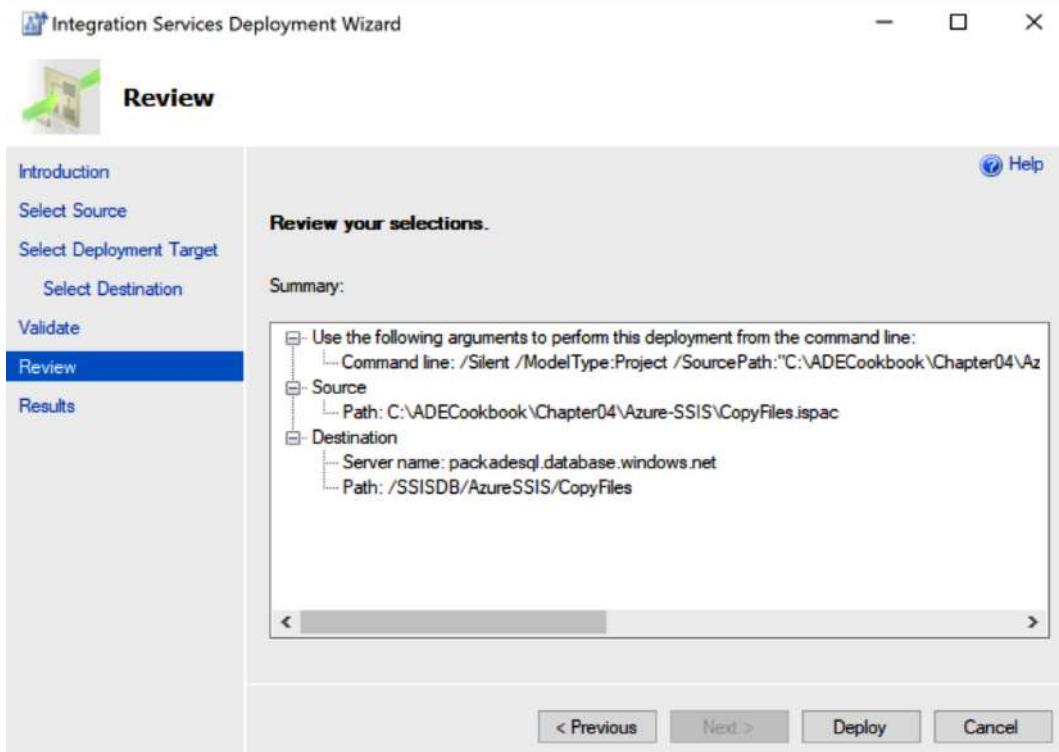


Figure 4.55 – Review and Deploy in the Integration Services Deployment Wizard

After a successful deployment, close the wizard.

11. In the **Object Explorer** window in SSMS, refresh the AzureSSIS folder. Observe that the CopyFiles package is now listed under the AzureSSIS | Projects folder.

We'll now create an Azure Data Factory pipeline to execute the package we deployed to the SSISDB catalog. To do this, switch to the Azure portal. Open the Data Factory **Author** tab. Create a new pipeline called Pipeline-AzureSSIS-IR. Drag and drop the **Execute SSIS package** activity from the **Activities** section to the **General** section. In the **General** tab, name the activity **Execute CopyFiles SSIS Package**. Switch to the **Settings** tab. Select **AzureSSISIR** in the **Azure-SSIS IR** drop-down list. Set the package location as **SSISDB**. Select **AzureSSIS** from the **Folder** drop-down list. If you don't see the folder name in the drop-down list, click **Refresh**. Select **CopyFiles** in the **Project** drop-down list. Select **CopyFiles.dtsx** in the **Package** drop-down list:

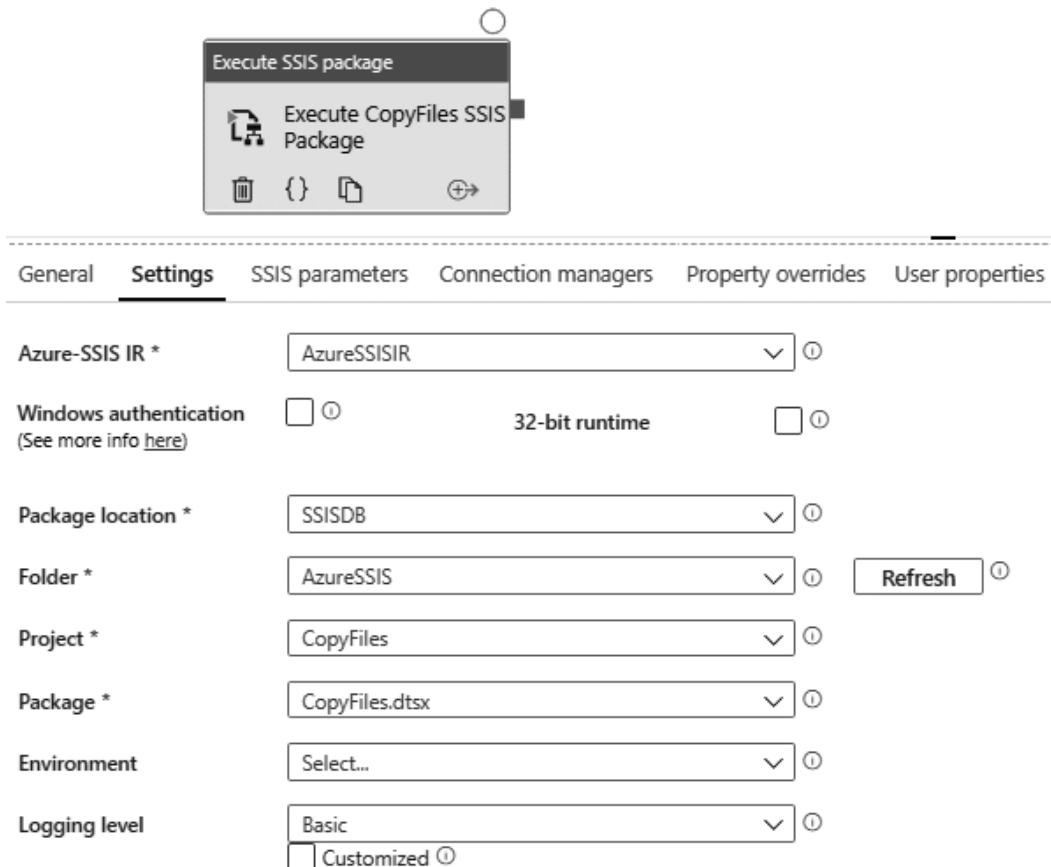


Figure 4.56 – Configuring the Execute SSIS package activity using the Settings tab

12. Switch to the **SSIS parameters** tab. Provide the **StorageAccountKey** value and the **StorageAccountName** value for the Azure Storage account:

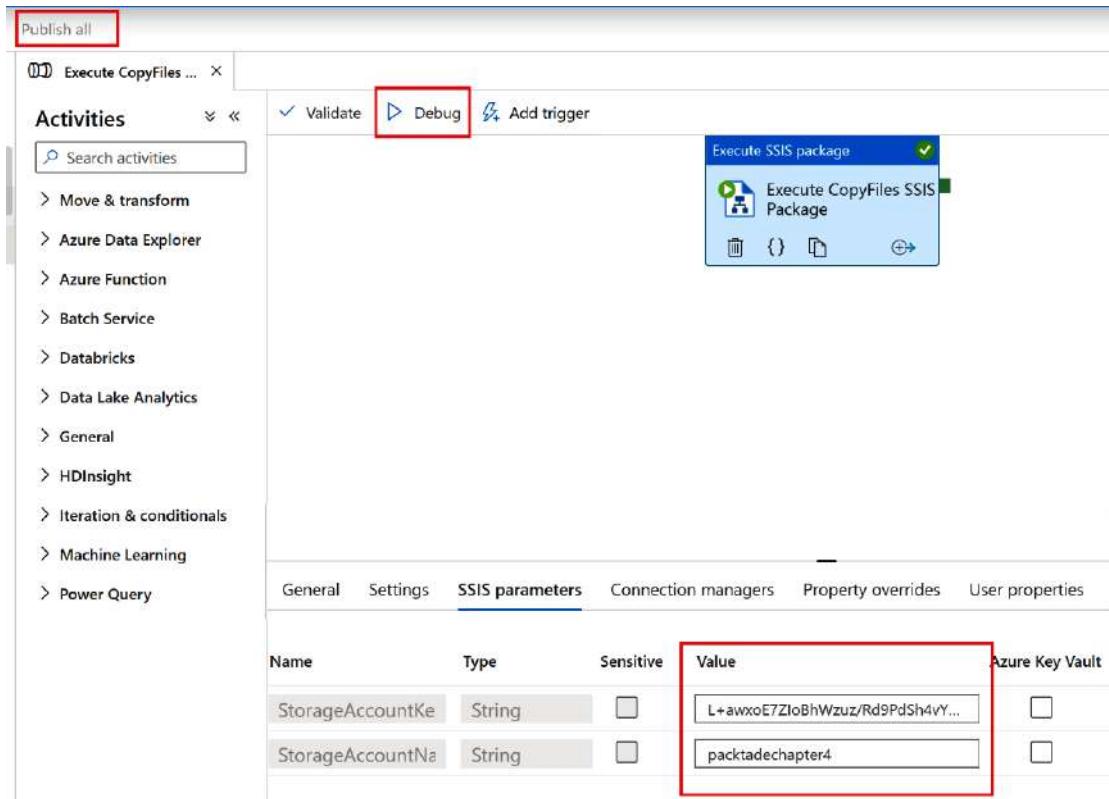


Figure 4.57 – SSIS parameters to configure the Execute SSIS package activity

Note

Make sure that the storage account you use has the `orders` container. You'll have to upload the `~/Chapter04/Azure-SSIS/Data/orders1.txt` file to the `orders/datain` folder.

13. Click **Publish all** to save your work. Click **Debug** to run the package. Once the package is complete, you should get an output similar to the one shown in the following screenshot:

The screenshot shows the 'Output' tab of a pipeline run in the Azure Data Factory portal. At the top, there are two tasks listed: 'Execute SSIS package' (status: succeeded) and 'Execute CopyFiles SSIS Package' (status: succeeded). Below this, the 'Pipeline run ID' is displayed as **55d7092b-19c1-4a23-8400-e8be83a6f86d**. The main table lists the task details:

Name	Type	Run start	Duration	Status	Integration runtime
Execute CopyFiles SSIS Package	Execute SSIS...	2021-12-25T03:28:38.238	00:00:24	✓ Succeeded	AzureSSISIR (East US)

Figure 4.58 – Viewing the output

Observe that the IR used is AzureSSISIR.

Note

To stop the Azure - SSIS IR, navigate to **Data Factory | Manage | Integration runtimes**.

Hover the mouse over the Azure - SSIS IR and click the **Pause** button.

How it works...

The Azure-SSIS IR lets us move on-premises SSIS packages to Azure Data Factory with few or no changes. We performed the following steps to move the **CopyFiles** SSIS package to Azure with no code change:

1. We created an Azure SSIS IR in our **ADFPacktADE2** Data Factory account.
2. In this recipe, we decided to store the package (**CopyFiles**) in Azure SQL Database's SSISDB. So, while configuring the Azure SSIS IR, we selected the option to create the SSISDB in Azure SQL Database (`packadesql.database.windows.net`). The SSISDB was provisioned while configuring the Azure SSIS IR.
3. We had to upload the **CopyFiles** package to the SSISDB. So, we connected to `packadesql.database.windows.net` using SSMS, created a project and folder inside the SSISDB, and uploaded the **CopyFiles** package into the folder using the **CopyFiles.ispac** package file.

4. We created the Data Factory pipeline, added the **Execute SSIS package** task, and configured it to use the `CopyFiles` package stored in the SSISDB. We also passed the storage account name (**packtadechapter4**) and its key as a parameter to the SSIS package so that it could move the files to the storage account.
5. We verified the successful execution by clicking the **Debug** button to run the package once. You can also verify it to see whether the `Order1.txt` file was successfully uploaded to the `orders/dataout` folder in **packtadechapter4**.

5

Configuring and Securing Azure SQL Database

Azure SQL Database, a fundamental relational database as a service offered in Azure, acts as a source, destination, or even as an intermediate storage layer in data engineering pipelines. Azure SQL Database can be used to consolidate data coming from several relational data sources and build mini data warehouses or data marts. With the introduction of Hyperscale tier in Azure SQL Database, the capacity of Azure SQL Database has increased leaps and bounds too. Securing Azure SQL Database is also pivotal in protecting access to the database. Having a strong understanding of Azure SQL Database's capabilities and security options is essential for any data engineer.

In this chapter, we will learn how to provision a serverless Azure SQL database, secure its connectivity to private links, integrate with Azure Key Vault to secure its credentials, configure a wake-up script to start a serverless Azure SQL database, and also configure the Hyperscale tier of Azure SQL Database.

In this chapter, we'll cover the following recipes:

- Provisioning and connecting to an Azure SQL database using PowerShell
- Implementing an Azure SQL Database elastic pool using PowerShell
- Configuring a virtual network and private endpoints for Azure SQL Database
- Configuring Azure Key Vault for Azure SQL Database
- Provisioning and configuring a wake-up script for a serverless SQL database
- Configuring the Hyperscale tier of Azure SQL Database

Technical requirements

For this chapter, the following are required:

- A Microsoft Azure subscription.
- PowerShell 7 or above.
- Microsoft Azure PowerShell – the installation instructions can be found at <https://docs.microsoft.com/en-us/powershell/azure/install-az-ps?view=azps-8.0.0>.

Provisioning and connecting to an Azure SQL database using PowerShell

In this recipe, we'll learn how to create and connect to an Azure SQL database instance. Azure SQL Database comes in three flavors: **standalone Azure SQL Database**, **Azure SQL Database elastic pools**, and **managed instances**. In this recipe, we'll create a standalone Azure SQL database.

Getting ready

In a new PowerShell window, execute the `Connect-AzAccount` command to log in to your Microsoft Azure account.

How to do it...

Let's begin by provisioning an Azure SQL database.

Provisioning an Azure SQL database

Execute the following steps to provision an Azure SQL database:

1. Execute the following PowerShell command to create a new resource group:

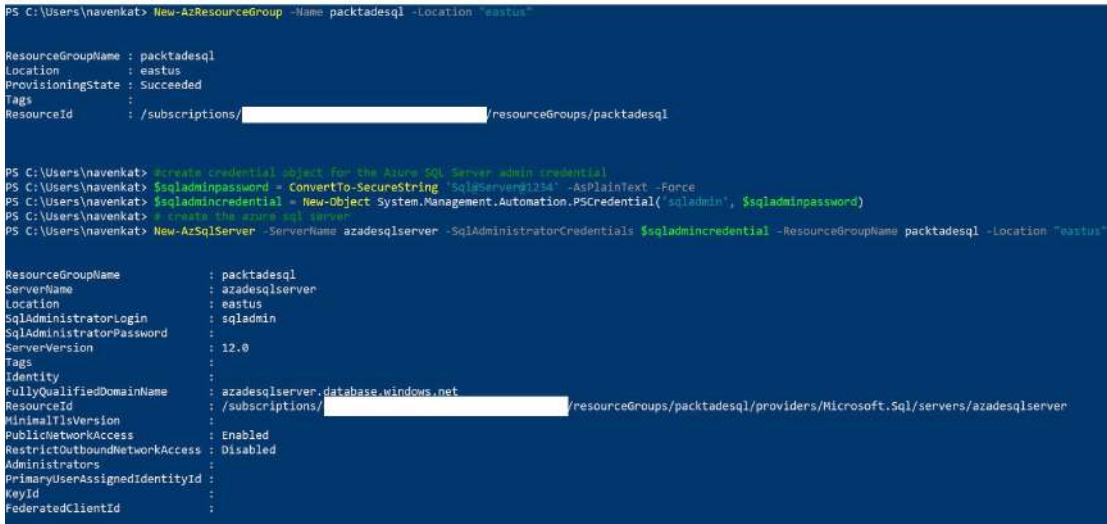
```
New-AzResourceGroup -Name packtadesql -Location "eastus"
```

2. Execute the following command to create a new Azure SQL server:

```
#create credential object for the Azure SQL Server admin  
credential  
  
$sqladminpassword = ConvertTo-SecureString 'Sql@  
Server@1234' -AsPlainText -Force  
  
$sqladmincredential = New-Object System.Management.  
Automation.PSCredential('sqladmin', $sqladminpassword)
```

```
# create the Azure SQL Server
New-AzSqlServer -ServerName azadesqlserver
-SqlAdministratorCredentials $sqladmincredential
-ResourceGroupName packtadesql -Location "eastus"
```

You should get an output similar to the one in the following screenshot:



```
PS C:\Users\navenkat> New-AzResourceGroup -Name packtadesql -Location "eastus"

ResourceGroupName : packtadesql
Location         : eastus
ProvisioningState : Succeeded
Tags              :
ResourceId       : /subscriptions/[REDACTED]/resourceGroups/packtadesql

PS C:\Users\navenkat> # create credential object for the Azure SQL server admin credential
PS C:\Users\navenkat> $sqladminpassword = ConvertTo-SecureString 'SqlServer@1234' -AsPlainText -Force
PS C:\Users\navenkat> $sqladmincredential = New-Object System.Management.Automation.PSCredential('sqladmin', $sqladminpassword)
PS C:\Users\navenkat> # create the azure sql server
PS C:\Users\navenkat> New-AzSqlServer -ServerName azadesqlserver -SqlAdministratorCredentials $sqladmincredential -ResourceGroupName packtadesql -Location "eastus"

ResourceGroupName      : packtadesql
ServerName            : azadesqlserver
Location              : eastus
SqlAdministratorLogin : sqladmin
SqlAdministratorPassword:
ServerVersion         : 12.0
Tags                 :
Identity              :
FullyQualifiedDomainName: azadesqlserver.database.windows.net
ResourceId            : /subscriptions/[REDACTED]/resourceGroups/packtadesql/providers/Microsoft.Sql/servers/azadesqlserver
MinIMALTlsVersion    :
PublicNetworkAccess   : Enabled
RestrictOutboundNetworkAccess: Disabled
Administrators        :
PrimaryUserAssignedIdentityId:
KeyId                :
FederatedClientId    :
```

Figure 5.1 – Creating a new Azure SQL server

3. Execute the following command to create a new Azure SQL database:

```
New-AzSqlDatabase -DatabaseName azadesqldb -Edition basic
-ServerName azadesqlserver -ResourceGroupName packtadesql
```

You should get an output similar to the one shown in the following screenshot:

```
PS C:\Users\naveenkat> New-AzSqlDatabase -DatabaseName azadesqldb -Edition basic -ServerName azadesqlserver -ResourceGroupName packtadesql
WARNING: Upcoming breaking changes in the cmdlet 'New-AzSqlDatabase':
- The output type 'Microsoft.Azure.Commands.Sql.Database.Model.AzureSqlDatabaseModel' is changing
- The following properties in the output type are being deprecated : 'BackupStorageRedundancy'
- The following properties are being added to the output type : 'CurrentBackupStorageRedundancy' 'RequestedBackupStorageRedundancy'
- The change is expected to take effect from the version : '3.0.0'
Note : Go to https://aka.ms/zps-changewarnings for steps to suppress this breaking change warning, and other information on breaking changes in Azure PowerShell.

ResourceGroupName          : packtadesql
ServerName                 : azadesqlserver
DatabaseName               : azadesqldb
Location                   : eastus
DatabaseId                : d9e7d136-77cb-49f6-b577-040b18a8b92e
Edition                    : Basic
CollationName              : SQL_Latin1_General_CI_AS
CatalogCollation           :
MaxSizeBytes               : 2147483648
Status                     : Online
CreationDate               : 30/12/2021 10:48:40 pm
CurrentServiceObjectiveId : 00000000-0000-0000-000000000000
CurrentServiceObjectiveName : Basic
RequestedServiceObjectiveName: Basic
RequestedServiceObjectiveId : Basic
ElasticPoolName            :
EarliestRestoreDate        :
Tags                       :
ResourceId                : /subscriptions/[REDACTED]/resourceGroups/packtadesql/providers/Microsoft.Sql/servers/azadesqlserver/databases/azadesqldb
CreateMode                 :
ReadScale                  : Disabled
ZoneRedundant               : False
Capacity                   : 5
Family                      :
SKUName                    : Basic
LicenseType                :
AutoPauseDelayInMinutes    :
MinimumCapacity             :
ReplicaType                :
HighAvailabilityReplicaCount: 1
CurrentBackupStorageRedundancy: Geo
RequestedBackupStorageRedundancy: Geo
SecondaryType               :
MaintenanceConfigurationId: /subscriptions/[REDACTED]/providers/Microsoft.Maintenance/publicMaintenanceConfigurations/SQL_Default
EnableLedger                : False
```

Figure 5.2 – Creating a new Azure SQL database

Connecting to an Azure SQL database

To connect to an Azure SQL database, let's first whitelist the IP address in the Azure SQL Server firewall:

1. Execute the following command to whitelist the public IP address of the machine to connect to an Azure SQL database (this recipe assumes that you are connecting from your local system. To connect from a system other than your local system, change the IP address in the following command). Execute the following command in the PowerShell window to whitelist the machine's public IP address in the Azure SQL Server firewall:

```
$clientip = (Invoke-RestMethod -Uri https://ipinfo.io/json).ip
New-AzSqlServerFirewallRule -FirewallRuleName "home"
-StartIpAddress $clientip -EndIpAddress $clientip
-ServerName azadesqlserver -ResourceGroupName packtadesql
```

You will get an output similar to the one shown in the following screenshot:

```
PS C:\Users\naveenkat>
PS C:\Users\naveenkat> $clientip = (Invoke-RestMethod -Uri https://ipinfo.io/json).ip
PS C:\Users\naveenkat> New-AzSqlServerFirewallRule -FirewallRuleName "home" -StartIpAddress $clientip -EndIpAddress $clientip -ServerName azadesqlserver -ResourceGroupName packtadesql

ResourceGroupName : packtadesql
ServerName       : azadesqlserver
StartIpAddress   : 116.89.64.165
EndIpAddress     : 116.89.64.165
FirewallRuleName : home
```

Figure 5.3 – Creating a new Azure SQL Server firewall rule

2. Execute the following command to connect to an Azure SQL database from SQLCMD (SQLCMD comes with the SQL Server installation, or you can download the SQLCMD utility from <https://docs.microsoft.com/en-us/sql/tools/sqlcmd-utility?view=sql-server-ver15>):

```
sqlcmd -S "azadesqlserver.database.windows.net" -U  
sqladmin -P "Sql@Server@1234" -d azadesqldb -Q "Select  
name from sys.databases"
```

Here's the output:

```
PS C:\Users\naveenkat> sqlcmd -S "azadesqlserver.database.windows.net" -U sqladmin -P "Sql@Server@1234" -d azadesqldb -Q "Select name from sys.databases"  
name  
-----  
master  
azadesqldb  
  
(2 rows affected)  
PS C:\Users\naveenkat>
```

Figure 5.4 – Connecting to an Azure SQL database

How it works...

We first execute the `New-AzSqlServer` command to provision a new Azure SQL Server. The command accepts the server name, location, resource group, and login credentials.

An Azure SQL Server, unlike an on-premises SQL Server, is not a physical machine or a **virtual machine (VM)** that is accessible to customers.

We then execute the `New-AzSQLDatabase` command to create an Azure SQL database. This command accepts the database name, the Azure SQL Server name, the resource group, and the edition. There are multiple SQL database editions to choose from based on the application workload. However, for the sake of this demo, we will create a basic edition.

To connect to an Azure SQL database, we first need to whitelist the machine's IP address in the Azure SQL Server firewall. Only whitelisted IPs are allowed to connect to the database.

To whitelist the client's public IP, we use the `New-AzSqlServerFirewallRule` command. This command accepts the server name, resource group, and start and end IPs. We can whitelist either a single IP address or a range of IP addresses.

We can connect to an Azure SQL database from **SQL Server Management Studio (SSMS)**, SQLCMD, or Azure Data Studio, or with a programming language using the appropriate SQL Server drivers. When connecting to an Azure SQL database, we need to specify the server name as `azuresqlservername.database.windows.net`, and then specify the Azure SQL database to connect to.

Implementing an Azure SQL Database elastic pool using PowerShell

An elastic pool is a cost-effective mechanism to group single Azure SQL databases with varying peak usage times. For example, consider 20 different SQL databases with varying usage patterns, each S3 Standard storage class requiring 100 **database throughput units (DTUs)** to run. We need to pay for 100 DTUs separately. However, we can group all of them in an elastic pool of S3 Standard storage classes. In this case, we only need to pay for elastic pool pricing and not for each individual SQL database.

In this recipe, we'll create an elastic pool of multiple single Azure databases.

Getting ready

In a new PowerShell window, execute the `Connect-AzAccount` command and follow the steps to log in to your Azure account.

How to do it...

The steps for this recipe are as follows:

1. Execute the following query on an Azure SQL Server:

```
#create credential object for the Azure SQL Server admin
credential
$sqladminpassword = ConvertTo-SecureString 'Sql@
Server@1234' -AsPlainText -Force
$sqladmincredential = New-Object System.Management.
Automation.PSCredential('sqladmin', $sqladminpassword)
#create the Azure SQL Server
New-AzSqlServer -ServerName azadesqlserver
-SqlAdministratorCredentials $sqladmincredential
-Location "eastus" -ResourceGroupName packtadesql
#Execute the following query to create an elastic pool.
#Create an elastic pool
New-AzSqlElasticPool -ElasticPoolName adepool -ServerName
azadesqlserver -Edition standard -Dtu 100 -DatabaseDtuMin
20 -DatabaseDtuMax 100 -ResourceGroupName packtadesql
```

You should get an output similar to the one shown in the following screenshot:

The screenshot shows a PowerShell session with the following commands and output:

```
PS C:\Users\navenkat> #Create a credential object for the Azure SQL Server admin credential
PS C:\Users\navenkat> $sqladminpassword = ConvertTo-SecureString "SqlAdmin@1234" -AsPlainText -Force
PS C:\Users\navenkat> $sqladmincredential = New-Object System.Management.Automation.PSCredential('sa@azadesqlserver', $sqladminpassword)
PS C:\Users\navenkat> # Create the Azure SQL server
PS C:\Users\navenkat> New-AzSqlServer -ServerName azadesqlserver -SqlAdministratorCredential $sqladmincredential -Location "eastus" -ResourceGroupName packtadesql

ResourceGroupName          : packtadesql
ServerName                 : azadesqlserver
Location                   : eastus
SqlAdministratorLogin     : sa@azadesqlserver
SqlAdministratorPassword   :
ServerVersion               : 12.0
Tags                       :
Identity                   :
FullyQualifiedDomainName   : azadesqlserver.database.windows.net
ResourceId                  : /subscriptions/[REDACTED]/resourceGroups/packtadesql/providers/Microsoft.Sql/servers/azadesqlserver
AzureSqlTlsversion         : 1.3
PublicNetworkAccess         : Enabled
RestrictOutboundNetworkAccess: Disabled
Administrators              :
PrimaryUserAssignedIdentityId:
KeyId                      :
FederatedClientId          :

PS C:\Users\navenkat> #Execute the following query to create an elastic pool.
PS C:\Users\navenkat> #Create an elastic pool
PS C:\Users\navenkat> New-AzSqlElasticPool -ElasticPoolName adepool -ServerName azadesqlserver -Edition standard -Dtu 100 -DatabaseDtuMin 20 -DatabaseDtuMax 100 -ResourceGroupName packtadesql

ResourceId      : /subscriptions/[REDACTED]/resourceGroups/packtadesql/providers/Microsoft.Sql/servers/azadesqlserver/elasticPools/adepool
ResourceGroupName : packtadesql
ServerName       : azadesqlserver
ElasticPoolName  : adepool
Location         : eastus
CreationDate    : 30/12/2021 11:33:47 pm
State            : Ready
Edition          : Standard
SkuName          : StandardPool
DTU              : 100
DatabaseDtuMax  : 100
DatabaseDtuMin  : 20
Capacity          : 100
DatabaseCapacityMin: 20
DatabaseCapacityMax: 100
Family           :
MaxSizeBytes    : 107374182400
StorageMB        : 102400
Tags             :
ZoneRedundant    : False
MaintenanceConfigurationId: /subscriptions/[REDACTED]/providers/Microsoft.Maintenance/publicMaintenanceConfigurations/SQL_Default
LicenseType      :
```

Figure 5.5 – Creating a new Azure elastic pool

2. Execute the following query to create and add an Azure SQL database to an elastic pool:

```
New-AzSqlDatabase -DatabaseName azadedb1 -ElasticPoolName adepool -ServerName azadesqlserver -ResourceGroupName packtadesql
```

You should get an output similar to the one shown in the following screenshot:

```
PS C:\Users\navenkat> New-AzSqlDatabase -DatabaseName azadedb1 -ElasticPoolName adepool -ServerName azadesqlserver -ResourceGroupName packtadesql
WARNING: Upcoming breaking changes in the cmdlet 'New-AzSqlDatabase':
- The output type 'Microsoft.Azure.Commands.Sql.Database.Model.AzureSqlDatabaseModel' is changing
- The following properties in the output type are being deprecated : 'BackupStorageRedundancy'
- The following properties are being added to the output type : 'CurrentBackupStorageRedundancy' 'RequestedBackupStorageRedundancy'
- The change is expected to take effect from the version : '3.0.0'
Note : Go to https://aka.ms/azps-changewarnings for steps to suppress this breaking change warning, and other information on breaking changes in Azure PowerShell.

ResourceGroupName          : packtadesql
ServerName                 : azadesqlserver
DatabaseName               : azadedb1
Location                   : eastus
DatabaseId                : a5a3c58d-c7d3-477e-a55e-67d5ff7fd3e2
Edition                    : Standard
CollationName              : SQL_Latin1_General_CI_AS
CatalogCollation           :
MaxSizeBytes               : 268435456000
Status                     : Online
CreationDate               : 30/12/2021 11:38:54 pm
CurrentServiceObjectiveId : 00000000-0000-0000-0000-000000000000
CurrentServiceObjectiveName : ElasticPool
RequestedServiceObjectiveName : ElasticPool
RequestedServiceObjectiveId :
ElasticPoolName            :
EarliestRestoreDate        :
Tags                       :
ResourceId                 : /subscriptions/[REDACTED]/resourceGroups/packtadesql/providers/Microsoft.Sql/servers/azadesqlserver/databases/azadedb1
CreateMode                 : Enabled
ReadScale                  : Disabled
ZoneRedundant              : False
Capacity                   : 0
Family                     :
SkuName                    : ElasticPool
LicenseType                :
AutoPauseDelayInMinutes    :
MinimumCapacity             :
ReadReplicaCount           :
HighAvailabilityReplicaCount :
CurrentBackupStorageRedundancy : Geo
RequestedBackupStorageRedundancy : Geo
SecondaryType               :
MaintenanceConfigurationId : /subscriptions/[REDACTED]/providers/Microsoft.Maintenance/publicMaintenanceConfigurations/SQL_Default
EnabledLedger               : False

PS C:\Users\navenkat>
```

Figure 5.6 – Creating a new SQL database in an elastic pool

3. Execute the following query to create a new Azure SQL database outside of the elastic pool:

```
New-AzSqlDatabase -DatabaseName azadedb2 -Edition Standard -RequestedServiceObjectiveName S3 -ServerName azadesqlserver -ResourceGroupName packtadesql
```

You should get an output similar to the one shown in the following screenshot:

The screenshot shows a PowerShell window with the following command and its output:

```
PS C:\Users\navenvkat> New-AzSqlDatabase -DatabaseName azadedb2 -Edition Standard -RequestedServiceObjectiveName S3 -ServerName azadesqlserver -ResourceGroupName packtadesql
```

Output (redacted sensitive information):

```
WARNING: Upcoming breaking changes in the cmdlet 'New-AzSqlDatabase' :
- The output type 'Microsoft.Azure.Commands.Sql.Database.Model.AzureSqlDatabaseModel' is changing
- The following properties in the output type are being deprecated : 'BackupStorageRedundancy'
- The following properties are being added to the output type : 'CurrentBackupStorageRedundancy' 'RequestedBackupStorageRedundancy'
- The change is expected to take effect from the version : 3.0.0
Note : Go to https://aka.ms/azps-changewarnings for steps to suppress this breaking change warning, and other information on breaking changes in Azure PowerShell.
```

Property	Value
ResourceGroupName	: packtadesql
ServerName	: azadesqlserver
DatabaseName	: azadedb2
Location	: eastus
DatabaseId	: df4c6cb9-ccb9-45e9-bcef-64e132cd63b
Edition	: Standard
CollationName	: SQL_Latin1_General_CI_AS
CatalogCollation	:
MaxSizeabytes	: 268435456000
Status	: Online
CreationDate	: 30/12/2021 11:42:46 pm
CurrentServiceObjectiveId	: 00000000-0000-0000-0000-000000000000
CurrentServiceObjectiveName	: S3
RequestedServiceObjectiveName	: S3
RequestedServiceObjectiveId	:
ElasticPoolName	:
EarliestRestoreDate	:
Tags	:
ResourceId	: /subscriptions/[REDACTED]/resourceGroups/packtadesql/providers/Microsoft.Sql/servers/azadesqlserver/databases/azadedb2
CreateMode	:
ReadScale	: Disabled
ZoneRedundant	: False
Capacity	: 100
Family	:
SkuName	: Standard
Licensetype	:
AutoPauseDelayInMinutes	:
MinimumCapacity	:
ReadReplicaCount	:
HighAvailabilityReplicaCount	:
CurrentBackupStorageRedundancy	: Geo
RequestedBackupStorageRedundancy	: Geo
SecondaryType	:
MaintenanceConfigurationId	: /subscriptions/[REDACTED]/providers/Microsoft.Maintenance/publicMaintenanceConfigurations/SQL_Default
EnableLedger	: False

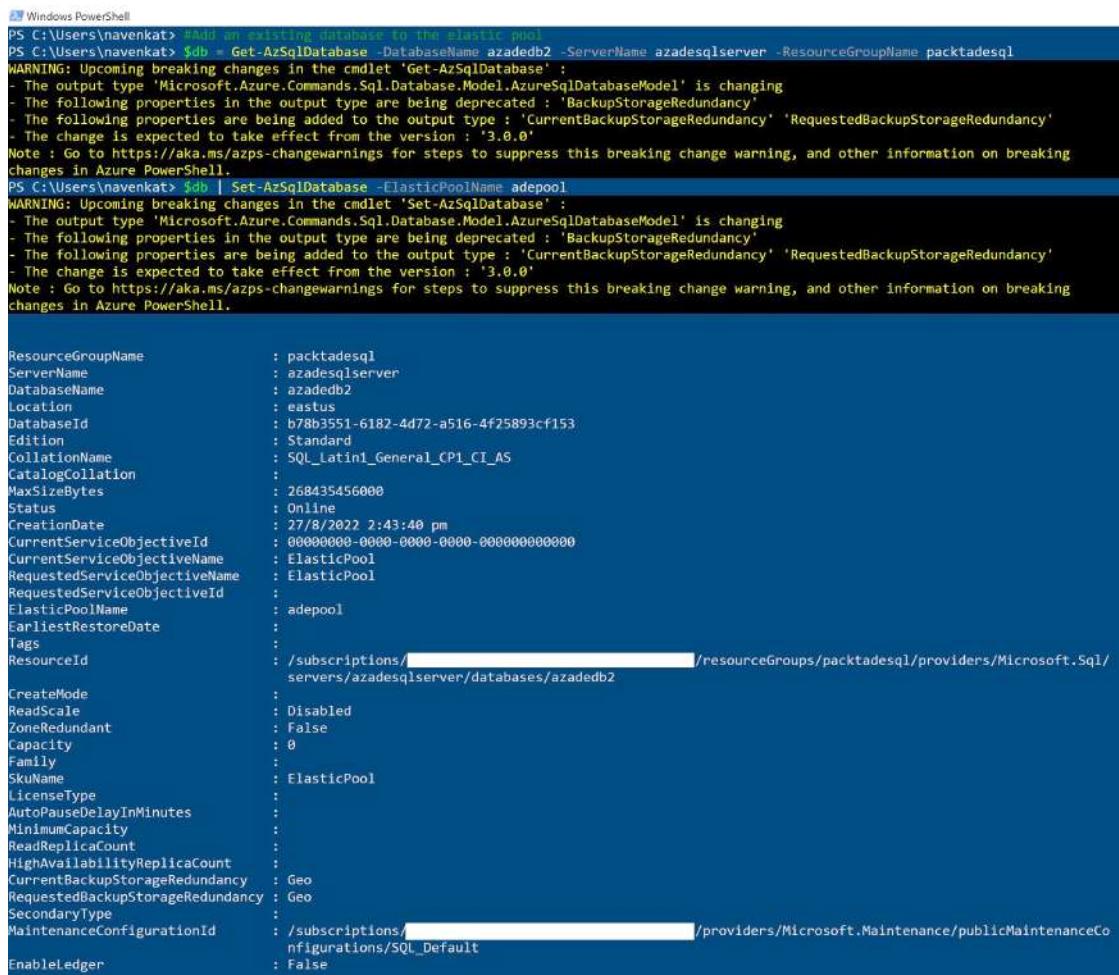
```
PS C:\Users\navenvkat>
```

Figure 5.7 – Creating a new SQL database

4. Execute the following query to add the azadedb2 database created in the preceding step to the elastic pool:

```
#Add an existing database to the elastic pool
$db = Get-AzSqlDatabase -DatabaseName azadedb2
-ServerName azadesqlserver -ResourceGroupName packtadesql
$db | Set-AzSqlDatabase -ElasticPoolName adepool
```

You should get an output similar to the one shown in the following screenshot:



```

PS C:\Users\navenkat> #Add an existing database to the elastic pool
PS C:\Users\navenkat> $db = Get-AzSqlDatabase -DatabaseName azadedb2 -ServerName azadesqlserver -ResourceGroupName packtadesql
WARNING: Upcoming breaking changes in the cmdlet 'Get-AzSqlDatabase':
- The output type 'Microsoft.Azure.Commands.Sql.Database.Model.AzureSqlDatabaseModel' is changing
- The following properties in the output type are being deprecated : 'BackupStorageRedundancy'
- The change is expected to take effect from the version : '3.0.0'
Note : Go to https://aka.ms/azps-changewarnings for steps to suppress this breaking change warning, and other information on breaking
changes in Azure PowerShell.
PS C:\Users\navenkat> $db | Set-AzSqlDatabase -ElasticPoolName adepool
WARNING: Upcoming breaking changes in the cmdlet 'Set-AzSqlDatabase':
- The output type 'Microsoft.Azure.Commands.Sql.Database.Model.AzureSqlDatabaseModel' is changing
- The following properties in the output type are being deprecated : 'BackupStorageRedundancy'
- The following properties are being added to the output type : 'CurrentBackupStorageRedundancy' 'RequestedBackupStorageRedundancy'
- The change is expected to take effect from the version : '3.0.0'
Note : Go to https://aka.ms/azps-changewarnings for steps to suppress this breaking change warning, and other information on breaking
changes in Azure PowerShell.

ResourceGroupName          : packtadesql
ServerName                 : azadesqlserver
DatabaseName               : azadedb2
Location                   : eastus
DatabaseId                : b78b3551-6182-4d72-a516-4f25893cf153
Edition                    : Standard
CollationName              : SQL_Latin1_General_CI_AS
CatalogCollation           :
MaxSizeBytes               : 268435456000
Status                     : Online
CreationDate               : 27/8/2022 2:43:40 pm
CurrentServiceObjectiveId : 00000000-0000-0000-0000-000000000000
CurrentServiceObjectiveName:
RequestedServiceObjectiveName:
RequestedServiceObjectiveId:
ElasticPoolName            : adepool
EarliestRestoreDate        :
Tags                       :
ResourceId                : /subscriptions/[REDACTED]/resourceGroups/packtadesql/providers/Microsoft.Sql/
CreateMode                 :
ReadScale                  : Disabled
ZoneRedundant              : False
Capacity                   : 0
Family                     :
SkuName                    : ElasticPool
LicenseType                :
AutoPauseDelayInMinutes    :
MinimumCapacity             :
ReadReplicaCount           :
HighAvailabilityReplicaCount:
CurrentBackupStorageRedundancy: Geo
RequestedBackupStorageRedundancy: Geo
SecondaryType               :
MaintenanceConfigurationId: /subscriptions/[REDACTED]/providers/Microsoft.Maintenance/publicMaintenanceCo
EnableLedger                : False

```

Figure 5.8 – Adding an existing SQL database to an elastic pool

5. To verify this in the Azure portal, log in with your Azure account. Navigate to **All resources** | **azadesqlserver** | **SQL elastic pools** | **adepool** | **Configure** and click on the **Databases** tab:

The screenshot shows the Azure portal interface for managing an elastic pool. The left sidebar has a 'Configure' tab highlighted with a red box. The main content area is titled 'adepool (azadesqlserver/adepool) | Configure'. Under 'Pool settings', the 'Databases' tab is selected (also highlighted with a red box). A table lists databases ready to be added to the pool, showing 'Database name', 'Pricing tier', and 'Data space'. Two databases, 'azadedb1' and 'azadedb2', are listed with 0 eDTU usage and 20 MB data space.

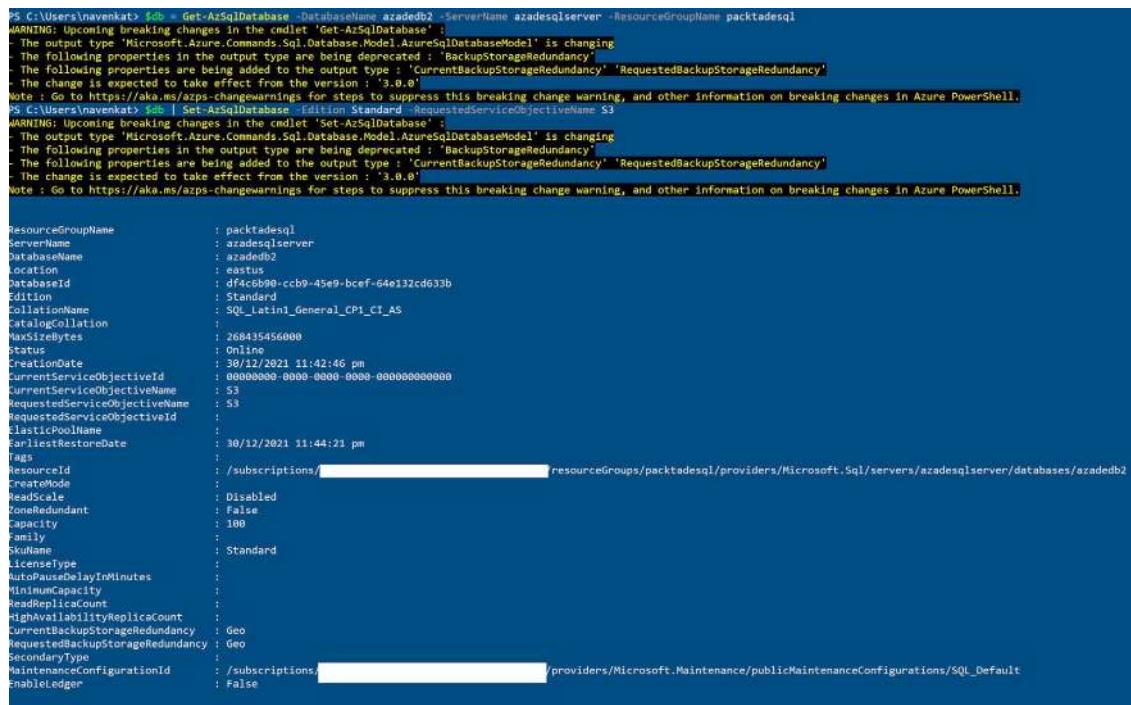
Database name	Pricing tier	Data space
azadedb1	0	20 MB
azadedb2	0	20.06 MB

Figure 5.9 – Viewing the elastic pool in the Azure portal

6. Execute the following command to remove an Azure SQL database from an elastic pool. To move a database out of an elastic pool, we need to set the edition and the service objective explicitly:

```
#remove a database from an elastic pool
$db = Get-AzSqlDatabase -DatabaseName azadedb2
-ServerName azadesqlserver -ResourceGroupName packtadesql
$db | Set-AzSqlDatabase -Edition Standard
-RequestedServiceObjectiveName S3
```

You should get an output similar to the one shown in the following screenshot:



```

PS C:\Users\havenvkat> $db = Get-AzSqlDatabase -DatabaseName azadedb2 -ServerName azadesqlserver -ResourceGroupName packtadesql
WARNING: Upcoming breaking changes in the cmdlet 'Get-AzSqlDatabase':
- The output type 'Microsoft.Azure.Commands.Sql.Database.Model.AzureSqlDatabaseModel' is changing
- The following properties in the output type are being deprecated : 'BackupStorageRedundancy'
- The following properties are being added to the output type : 'CurrentBackupStorageRedundancy' 'RequestedBackupStorageRedundancy'
- The change is expected to take effect from the version : '3.0.0'
Note : Go to https://aka.ms/azps-changewarnings for steps to suppress this breaking change warning, and other information on breaking changes in Azure PowerShell.
PS C:\Users\havenvkat> Set-AzSqlDatabase -DatabaseName azadedb2 -ServiceObjectiveName S3
WARNING: Upcoming breaking changes in the cmdlet 'Set-AzSqlDatabase':
- The output type 'Microsoft.Azure.Commands.Sql.Database.Model.AzureSqlDatabaseModel' is changing
- The following properties in the output type are being deprecated : 'BackupStorageRedundancy'
- The following properties are being added to the output type : 'CurrentBackupStorageRedundancy' 'RequestedBackupStorageRedundancy'
- The change is expected to take effect from the version : '3.0.0'
Note : Go to https://aka.ms/azps-changewarnings for steps to suppress this breaking change warning, and other information on breaking changes in Azure PowerShell.

ResourceGroupName          : packtadesql
ServerName                 : azadesqlserver
DatabaseName               : azadedb2
Location                   : eastus
DatabaseId                : df4c8c90-ccb9-45e9-bcef-64e132cd633b
Edition                    : Standard
CollectionName             : SQL_Latin1_General_CI_AS
CatalogCollation          :
MaxSizeBytes               : 268435456000
Status                     : Online
CreationDate               : 30/12/2021 11:42:46 pm
CurrentServiceObjectiveId : 00000000-0000-0000-0000-000000000000
CurrentServiceObjectiveName: S3
RequestedServiceObjectiveName: S3
RequestedServiceObjectiveId : 00000000-0000-0000-0000-000000000000
ElasticPoolName            :
EarliestRestoresDate      : 30/12/2021 11:44:21 pm
Tags                       :
ResourceId                : /subscriptions/[REDACTED]/resourceGroups/packtadesql/providers/Microsoft.Sql/servers/azadesqlserver/databases/azadedb2
CreateMode                 :
ReadScale                  : Disabled
ZoneRedundant              : False
Capacity                  : 100
Family                     :
SkulName                  :
LicenseType                :
AutoPauseDelayInMinutes    :
MinimumCapacity            :
ReadReplicaCount           :
HighPriorityReplicaCount   :
CurrentBackupStorageRedundancy: Geo
RequestedBackupStorageRedundancy: Geo
SecondaryType              :
MaintenanceConfigurationId: /subscriptions/[REDACTED]/providers/Microsoft.Maintenance/publicMaintenanceConfigurations/SQL_Default
EnableLedge                : False

```

Figure 5.10 – Removing a SQL database from an elastic pool

7. Execute the following command to remove an elastic pool. An elastic pool has to be empty before it can be removed. Execute the following query to remove all the databases in an elastic pool:

```

# get elastic pool object
$epool = Get-AzSqlElasticPool -ElasticPoolName adepool
-ServerName azadesqlserver -ResourceGroupName packtadesql
# get all databases in an elastic pool
$epdbs = $epool | Get-AzSqlElasticPoolDatabase
# change the edition of all databases in an elastic pool
to standard S3
foreach($db in $epdbs) {
    $db | Set-AzSqlDatabase -Edition Standard
    -RequestedServiceObjectiveName S3
}
# Remove an elastic pool
$epool | Remove-AzSqlElasticPool

```

Note

The command sets the edition of the SQL databases to **Standard**. This is for demo purposes only. If this is to be done in production, modify the edition and the service objective accordingly.

How it works...

We create an elastic pool using the `New-AzSqlElasticPool` command. In addition to the parameters, such as the server name, resource group name, compute model, compute generation, and edition, which are the same as when we created a new Azure SQL database, we can also specify `DatabaseMinDtu` and `DatabaseMaxDtu`. `DatabaseMinDtu` specifies the minimum amount of DTUs that all the databases in an elastic pool can have. `DatabaseMaxDtu` is the maximum amount of DTUs that a database can consume in an elastic pool.

Similarly, for the vCore-based purchasing model, we can specify `DatabaseVCoreMin` and `DatabaseVCoreMax`.

To add a new database to an elastic pool, specify the elastic pool name at the time of database creation using the `New-AzSqlDatabase` command.

To add an existing database to an elastic pool, modify the database using `Set-AzSqlDatabase` to specify the elastic pool name.

To remove a database from an elastic pool, modify the database using the `Set-AzSqlDatabase` command to specify a database edition explicitly.

To remove an elastic pool, first, empty it by moving all of the databases out of the elastic pool, and then remove it using the `Remove-AzSqlElasticPool` command.

Configuring a virtual network and private endpoints for Azure SQL Database

Securing the connectivity to an Azure SQL database is important to limit the exposure of the database to external attacks such as **distributed denial-of-service (DDOS)** attacks and **SQL injection**. Using private endpoints for connecting to Azure SQL Database ensures that the database connectivity flows through Azure's backbone network and does not use the public internet. Placing the SQL endpoint behind a virtual network prevents Azure SQL Database from being exposed to a connection request from the public internet. In this recipe, we will explore how we can configure private endpoints using virtual networks for Azure SQL Database.

Getting ready

1. Log in to `portal.azure.com`.
2. Create an Azure SQL Database, as explained in the *Provisioning and connecting to an Azure SQL database using PowerShell* recipe in this chapter.

How to do it...

Perform the following steps to create an Azure SQL database with a private link and virtual network:

1. **Creating a virtual network:** Go to portal.azure.com and click **Create a Resource**. Search for **Virtual Network** and click **Create**, as shown in the following screenshot:

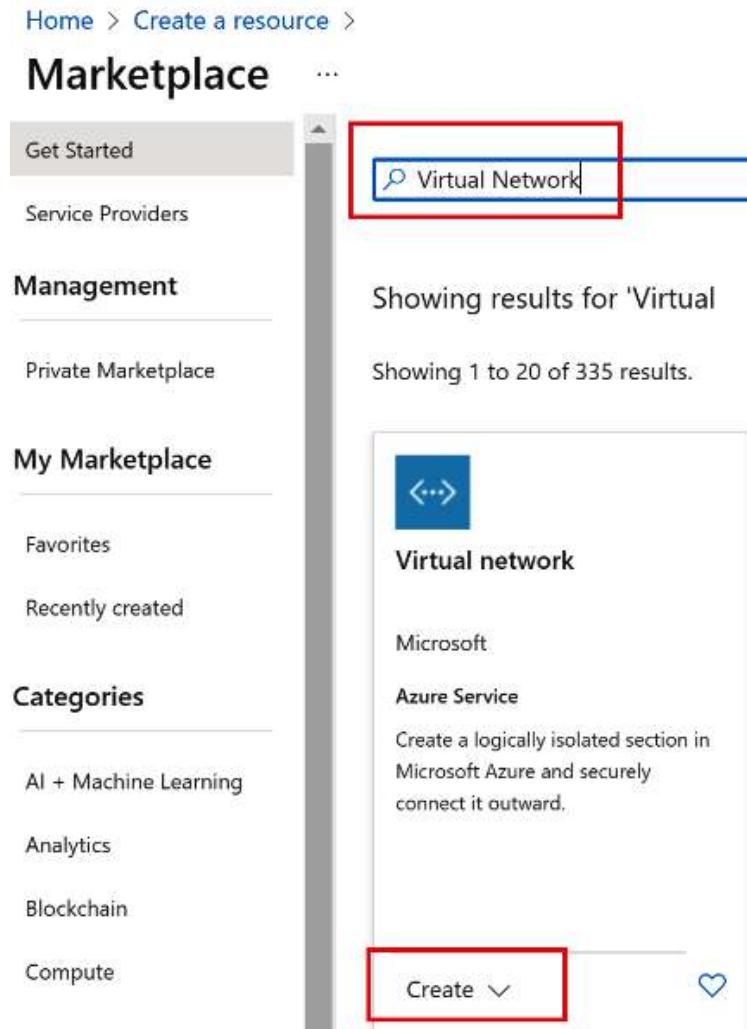


Figure 5.11 – Creating a virtual network

2. Provide the resource group name as `packtadesql` (create one if it doesn't exist using the **Create new** button). Provide the virtual network instance name as `packtadesqlvnet`, as shown in the following screenshot. Click **Review + create**:

Home > Create a resource > Marketplace >

Create virtual network

Basics IP Addresses Security Tags Review + create

Azure Virtual Network (VNet) is the fundamental building block for your private network in Azure. VNet enables many types of Azure resources, such as Azure Virtual Machines (VM), to securely communicate with each other, the internet, and on-premises networks. VNet is similar to a traditional network that you'd operate in your own data center, but brings with it additional benefits of Azure's infrastructure such as scale, availability, and isolation. [Learn more about virtual network](#)

Project details

Subscription * ⓘ Visual Studio Ultimate with MSDN

Resource group * ⓘ packtadesql

Instance details

Name * packtadesqlvnet

Region * East US

Review + create

< Previous

Next : IP Addresses >

Download a template for automation

Figure 5.12 – Create virtual network

3. Once the virtual network has been created, go to **All resources**. Find **azadesqlserver**. Click **azadesqlserver** and click **Firewalls and virtual networks**, as shown here:

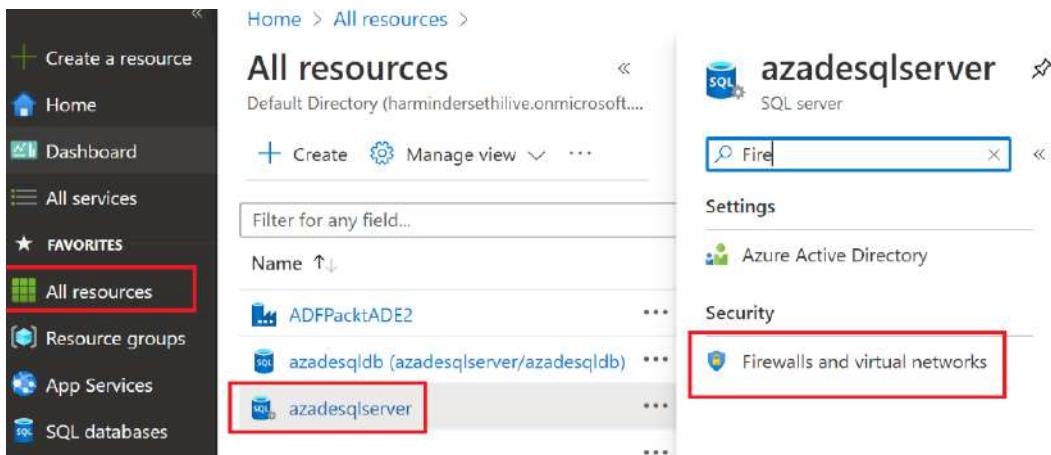


Figure 5.13 – Firewalls and virtual networks

4. Check the **Deny public network access** checkbox and click the **Save** button:

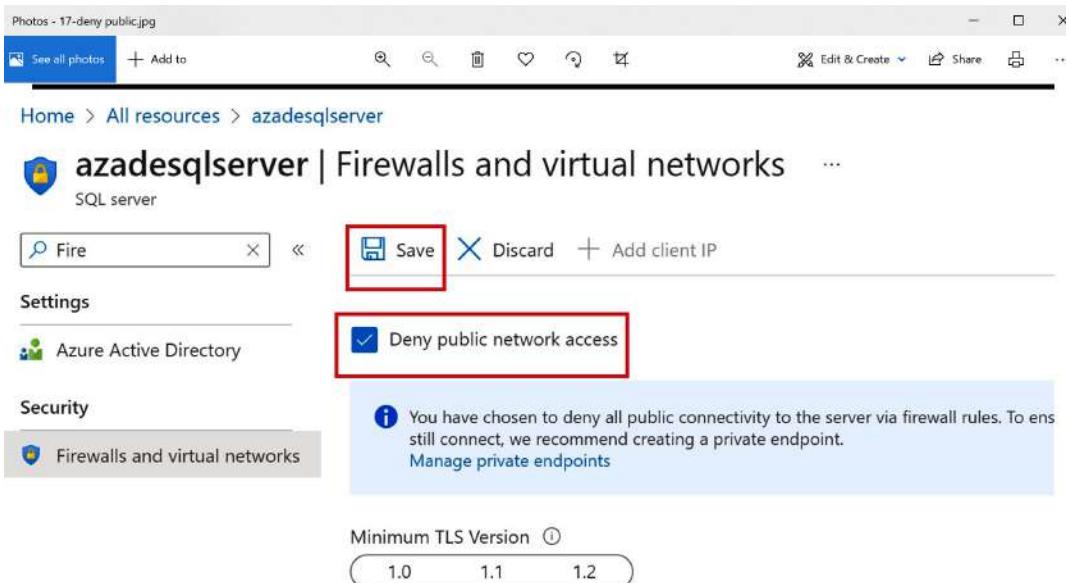


Figure 5.14 – Firewalls and virtual networks

- Now, on **azadesqlserver**, search for **Private endpoint** and click the **+ Private endpoint** button:

The screenshot shows the 'azadesqlserver | Private endpoint connections' blade. At the top, there's a search bar with 'Private' and several action buttons: '+ Private endpoint' (highlighted with a red box), 'Approve', 'Reject', 'Remove', and 'Refresh'. Below this, there's a 'Security' section with a red box around the 'Private endpoint connections' link. The main area is titled 'Private Endpoint Connection' and contains a description: 'Private endpoint connections allow connections from within a Virtual Network to Connections using these private endpoints specified below provide access to all'. There's a search bar ('Search...') and a dropdown showing '3 selected'. A table header includes 'Connection name', 'State', and 'Private en'. Below the table, a note says 'Click on add to create private endpoint'.

Figure 5.15 – Private endpoint

- Create a private endpoint named **pepsqlserver**. Ensure that the **Region** detail is the same as that of Azure SQL Database (**East US**):

Home > All resources > azadesqlserver >

Create a private endpoint ...

The screenshot shows the 'Create a private endpoint' blade. It has five tabs: **Basics** (selected), **Resource**, **Configuration**, **Tags**, and **Review + create**. A note says: 'Use private endpoints to privately connect to a service or resource. Your private endpoint must be in the same region as your virtual network, but can be in a different region from the private link resource that you are connecting to.' A 'Learn more' link is provided. The 'Project details' section includes 'Subscription *' (Visual Studio Ultimate with MSDN) and 'Resource group *' (packtadesql). The 'Instance details' section includes 'Name *' (pepsqlserver) and 'Region *' (East US). Both the 'Name' field and the 'Region' dropdown are highlighted with a red box.

Figure 5.16 – Create a private endpoint

7. On the **Resource** tab, set **Resource type** to **Microsoft.Sql/servers**, **Resource** to **azadesqlserver**, and **Target sub-resource** to **sqlServer**, as shown here:

Home > packtadesql > azadesqlserver >

Create a private endpoint

Basics **Resource** Configuration Tags Review + create

Private Link offers options to create private endpoints for different Azure resources, like your private link service, a SQL server, or an Azure storage account. Select which resource you would like to connect to using this private endpoint. [Learn more](#)

Connection method (1)

Connect to an Azure resource in my directory.
 Connect to an Azure resource by resource ID or alias.

Subscription * (1)

Visual Studio Ultimate with MSDN

Resource type * (1)

Microsoft.Sql/servers

azadesqlserver

sqlServer



Figure 5.17 – Resource type

8. Move to the **Configuration** tab. Set **Virtual network** to **packtadesqlvnet**, which we created in *step 1*:

[Home](#) > [packtadesql](#) > [azadesqlserver](#) >

Create a private endpoint

✓ Basics ✓ Resource Configuration Tags Review + create

Networking

To deploy the private endpoint, select a virtual network subnet. [Learn more](#)

Virtual network *

Subnet *

ⓘ If you have a network security group (NSG) enabled for the subnet above, it will be disabled for private endpoints on this subnet only. Other resources on the subnet will still have NSG enforcement.

Private DNS integration

To connect privately with your private endpoint, you need a DNS record. We recommend that you integrate your private endpoint with a private DNS zone. You can also utilize your own DNS servers or create DNS records using the host files on your virtual machines. [Learn more](#)

Integrate with private DNS zone Yes No

Configuration name	Subscription	Resource group	Private DNS zone
privatelink-database-windows-net	Visual Studio Ultimate with MSDN	<input type="text" value="packtadesql"/> 	(new) privatelink.database.windows.net

Figure 5.18 – Create a private endpoint

9. Hit **Review + create** and then the **Create** button to create the Azure SQL database with a private endpoint. With this, we have created a private endpoint to connect to Azure SQL Database.
10. To test it, let's try to connect to the database from our local machine. It is expected to fail, as it's not connected to the virtual network. Install SSMS and test the connectivity to `azadesqlserver.database.windows.net`. It fails with an error message, as shown in the following screenshot:



Figure 5.19 – Connection error

11. Create a new VM in the Azure portal. Ensure to allow remote desktop connection to the VM, as shown here:

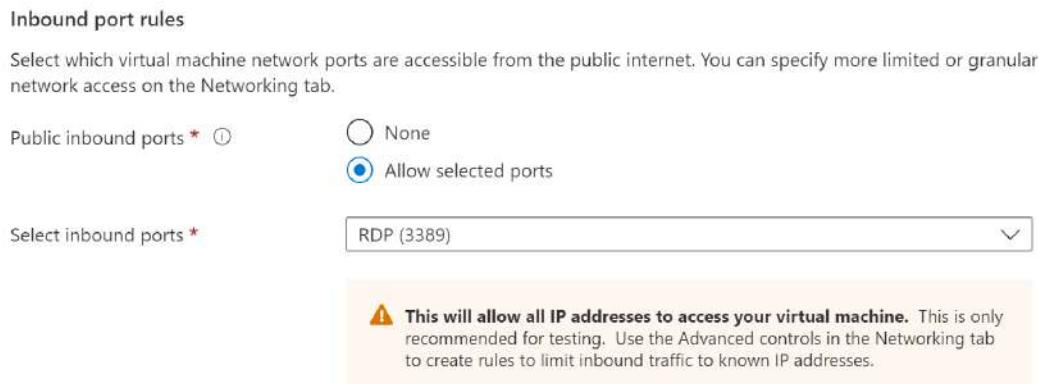


Figure 5.20 – Creating a new VM and allowing remote desktop connection

12. While creating the VM, under **Networking**, set **Virtual network** to **packtadesqlvnet**. A VM will be created inside the virtual network:

The screenshot shows the 'Create a virtual machine' wizard, step 3: Networking. The 'Networking' tab is selected. Under 'Network interface', it lists 'Virtual network' as 'packtadesqlvnet', 'Subnet' as 'default (10.0.0.0/24)', and 'Public IP' as '(new) SQLVM-ip'. Under 'NIC network security group', 'Basic' is selected.

Figure 5.21 – Configuring the VM to use the virtual network

13. Once the VM is created, open SSMS and test the connectivity to `azadesqlserver.database.windows.net`. It will successfully connect to the database, as it is now part of the virtual network.

How it works...

The objective of the recipe was to ensure that the database can only be connected using Azure's backbone network and no connection from the public internet is allowed. To achieve this, we performed the following:

1. We created a virtual network called **packtadesqlvnet**.
2. The private endpoint acts as a private tunnel between the client connection and the resource (Azure SQL Database). So, we created a private endpoint connection called **pepsqlserver** inside the virtual network, **packtadesqlvnet**. We configured the private endpoint, **pepsqlserver**, to connect to our Azure SQL database, **azadesqlserver**. This ensures that the endpoint can be used to connect to the database.
3. We shut down any public connections to the Azure SQL database, **azadesqlserver**, by setting **Deny public network access**. This ensures that no connection through a public network is possible to our Azure SQL database, **azadesqlserver**.
4. A private endpoint connection only works from devices joined to the Azure virtual network. We tested this by trying to connect from a public network (our machine), which failed. We then created a VM inside the virtual network (**packtadesqlvnet**) and tested the connectivity to the database, which was successful.

Configuring Azure Key Vault for Azure SQL Database

Azure SQL Database is encrypted at rest by default using Microsoft-managed keys. However, many customers prefer to encrypt Azure SQL Database using keys that are managed by them, as it offers more control over the encryption keys. This recipe will show how you can use customer-managed keys to encrypt Azure SQL Database by integrating it with Azure Key Vault.

Getting ready

Create an Azure SQL database, as explained in the *Provisioning and connecting to an Azure SQL database using PowerShell* recipe in this chapter.

How to do it...

Perform the following steps to configure Azure Key Vault:

1. Go to `portal.azure.com`, click **All resources**, and find the SQL server, **azadesqlserver**.
2. Find **Transparent data encryption** under **Security**. Click **Customer-managed key** and click **Change key**:

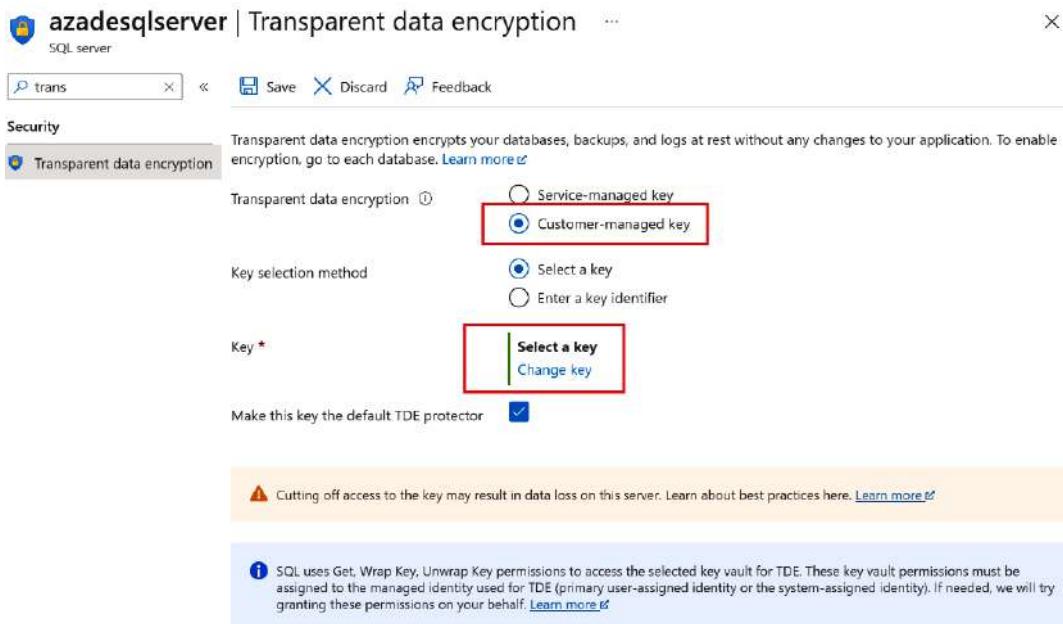


Figure 5.22 – Configuring Azure Key Vault

3. Set **Key store type** to **Key vault**. Click **Create new key vault**:

Home > All resources > azadesqlserver >

Select a key

Subscription * Visual Studio Ultimate with MSDN

Key store type Key vault Managed HSM

Key vault * Create new key vault

Key Create new key

Version Create new version

Figure 5.23 – Configuring Azure Key Vault

4. Provide azadekeyvault (or any new name) as the **Key vault name** and hit the **Review + create** button:

Home > All resources > azadesqlserver > Select a key >

Create a key vault ...

Basics Access policy Networking Tags Review + create

Azure Key Vault is a cloud service used to manage keys, secrets, and certificates. Key Vault eliminates the need for developers to store security information in their code. It allows you to centralize the storage of your application secrets which greatly reduces the chances that secrets may be leaked. Key Vault also allows you to securely store secrets and keys backed by Hardware Security Modules or HSMs. The HSMs used are Federal Information Processing Standards (FIPS) 140-2 Level 2 validated. In addition, key vault provides logs of all access and usage attempts of your secrets so you have a complete audit trail for compliance.

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 Ultimate with MSDN

Resource group * packtadesql [Create new](#)

Instance details	
Key vault name *	azadekeyvault
Region *	East US
Pricing tier *	Standard

Recovery options

Soft delete protection will automatically be enabled on this key vault. This feature allows you to recover or permanently delete a key vault and secrets for the duration of the retention period. This protection applies to the key vault and the secrets stored within the key vault.

To enforce a mandatory retention period and prevent the permanent deletion of key vaults or secrets prior to the retention period elapsing, you can turn on purge protection. When purge protection is enabled, secrets cannot be purged by users or by Microsoft.

[Review + create](#)

< Previous

Next : Access policy >

Figure 5.24 – New key vault creation

5. After the key vault is created in step 4, you will be taken back to the **Select a key** screen again. Click **Create new key**:

Home > All resources > azadesqlserver >

Select a key

Subscription *: Visual Studio Ultimate with MSDN

Key store type: Key vault Managed HSM

Key vault *: azadekeyvault
Create new key vault

Key:

Version:

Figure 5.25 – Create a new key

- Provide a key name. It is recommended to use a name that makes it easy to identify its purpose. Then, click **Create**:

Home > All resources > azadesqlserver > Select a key >

Create a key

Options: Generate

Name *: azadesqlserver

Key type: RSA EC

RSA key size: 2048 3072 4096

Set activation date:

Set expiration date:

Enabled: Yes No

Tags: 0 tags

Set key rotation policy (Preview): Not configured

Figure 5.26 – Create a new key

7. You will be taken back to the **Select a key** screen, but this time, all the values for **Key vault**, **Key**, and **Version** will automatically be filled in. Click the **Select** button:

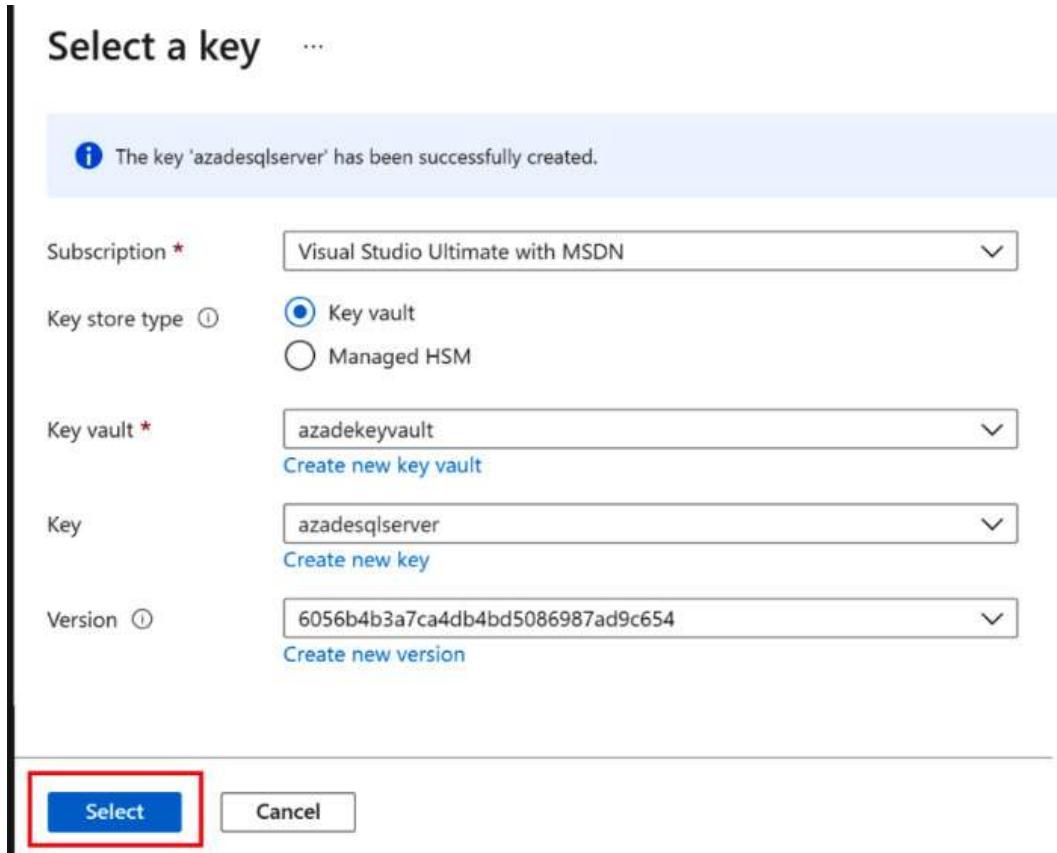


Figure 5.27 – Key selection completion

8. Check the **Auto-rotate key** checkbox. Hit the **Save** button to complete the configuration:

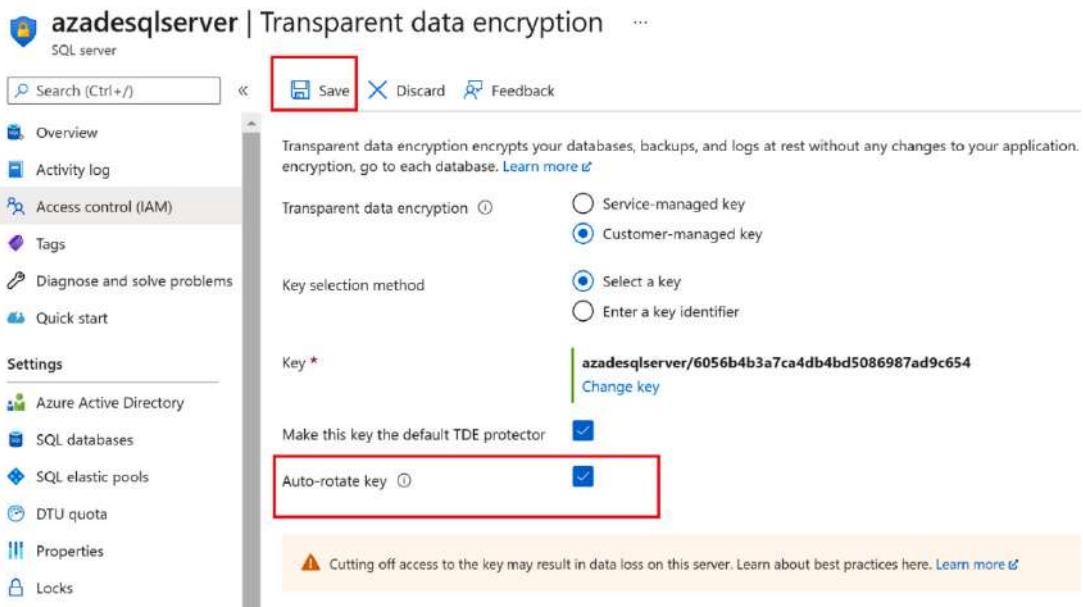


Figure 5.28 – Save the transparent data encryption configuration

How it works...

Turning on **Transparent data encryption** for Azure SQL Database using Azure Key Vault ensures that the database is always encrypted when at rest. By turning on the **Auto-rotate key** option while configuring the Key vault integration, we have ensured keys used for encryption are rotated automatically by Azure. Customers can still access the keys used in Key Vault using the Azure portal. This allows us to have the best of both worlds – enjoying the power of automation of the cloud while having control over your resources. During creation of Key Vault, Azure automatically grants permission for SQL Server to access Key Vault. You can verify this by going to **All resources** in the Azure portal, searching for **azadekeyvault**, and clicking **Access policies** under **Settings**:

The screenshot shows the 'azadekeyvault | Access policies' page in the Azure portal. The left sidebar has a 'Settings' section with 'Access policies' highlighted by a red box. The main area shows 'Enable Access to:' checkboxes for Azure VM deployment, Resource Manager template deployment, and Disk Encryption. The 'Permission model' is set to 'Vault access policy'. The 'Current Access Policies' table lists an 'APPLICATION' row for 'azadesqlserver' with three selected key permissions. There is also a 'USER' section below it.

Name	Email	Key Permissions	Secret Permission	Certifica
azadesqlserver		3 selected	0 selected	0 selec

Figure 5.29 – Verify permissions

Azure SQL Database will use its service account, **azadesqlserver**, access Key Vault using the permission granted, and perform operations such as encryption, decryption, and key rotation automatically with full transparency to users.

Provisioning and configuring a wake-up script for a serverless SQL database

Azure SQL Database is offered in two compute tiers: provisioned, and serverless. While a provisioned SQL database will be always running, a serverless database can pause when not in use and start again when connections flow in. Automatically pausing an Azure SQL database helps to save costs when the database is not being used.

While pausing the database is automatic, the database only starts when the first connection request comes in. Starting a database can take a few minutes of waiting time and cause inconvenience to customers. Configuring a wake-up script to start a paused database is a proactive method to reduce this waiting time. The following recipe will configure a serverless database and deploy a wake-up script to start the database at a specific schedule. At a high level, we will be doing the following:

1. Configuring the serverless compute tier for an existing database
2. Creating an **Azure Automation** account
3. Provisioning a PowerShell script to start the database
4. Scheduling the PowerShell script using a runbook in the Azure Automation account

Getting ready

Create an Azure SQL database, as explained in the *Provisioning and connecting to an Azure SQL database using PowerShell* section of this chapter.

How to do it...

Perform the following steps to configure a serverless database and schedule a wake-up script for the database:

1. Go to portal.azure.com, click **All resources**, and find the SQL database, **azadesqlDb**. Click **Compute + storage** under **Settings**. Pick **General Purpose (Scalable compute and storage options)** for **Service tier** and **Serverless** under **Computer tier**, as shown in the following screenshot:

The screenshot shows the Azure portal interface for managing an Azure SQL Database named 'azadesqlldb'. The left sidebar contains navigation links for Overview, Activity log, Tags, Diagnose and solve problems, Quick start, Query editor (preview), Power Platform (Power BI, Power Apps, Power Automate), Settings (selected), Connection strings, Properties, Locks, Data management (Replicas, Sync to other databases), Integrations (Stream analytics (preview), Add Azure Search), Security (Auditing), and Audit logs.

The main content area is titled 'Service and compute tier' and includes a note about selecting tiers based on workload needs. It features two sections: 'Service tier' (set to 'General Purpose (Scalable compute and storage options)') and 'Compute tier' (set to 'Serverless'). The 'Compute tier' section is highlighted with a red box.

The 'Compute Hardware' section allows configuring hardware based on workload requirements. It includes a 'Hardware Configuration' section with a 'Gen5' option (up to 40 vCores, up to 120 GB memory) and a 'Change configuration' link. Below it are sliders for 'Max vCores' (set to 2) and 'Min vCores' (set to 0.5 vCores). A summary bar at the bottom indicates '2.02 GB MIN MEMORY' and '3 GB MAX MEMORY'.

The 'Auto-pause delay' section provides information about pausing the database if inactive and includes a checked checkbox for 'Enable auto-pause'.

A blue 'Apply' button is located at the bottom of the configuration pane.

Figure 5.30 – Configure a serverless database for Azure SQL Database

2. Scroll down and make note of the **Auto-pause delay** duration of 1 hour and click the **Apply** button:

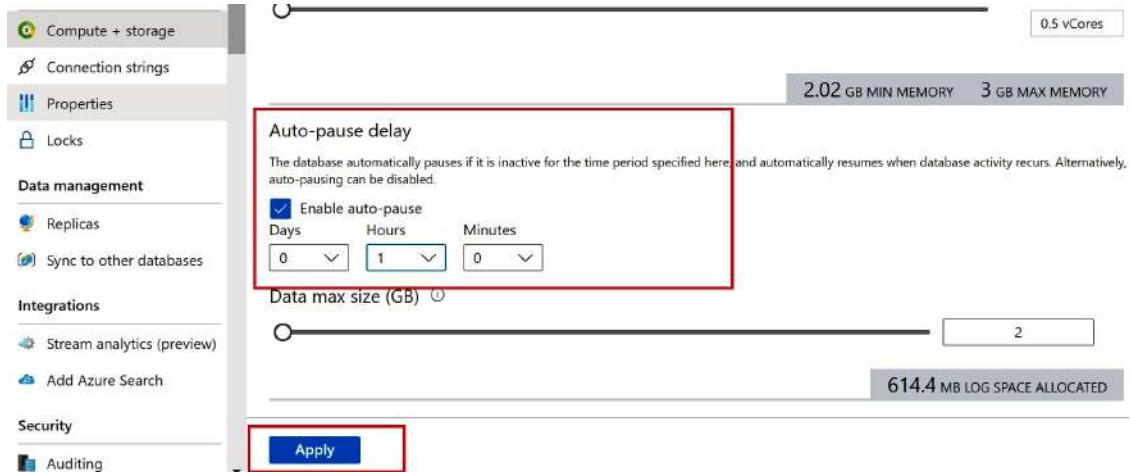


Figure 5.31 – Set auto-pause duration

3. To create an Automation account, go to portal.azure.com, click **Create a resource**, and search for **Automation Account**. Click **Create** for the **Automation** service listed by Microsoft:

Home > Create a resource >

Marketplace

...

Get Started

Service Providers

Management

Private Marketplace

My Marketplace

Favorites

Recently created

Categories

AI + Machine Learning

Analytics

Blockchain

Compute

Automation Account

Showing results for 'Automatio'

Showing 1 to 20 of 47 results.

Automation

Microsoft

Azure Service

Automate the management of your cloud and on-premises resources

Create

♥

Figure 5.32 – Create an Automation account

4. Provide the **Resource group** name as `packtadesql` (the same as the one given to Azure SQL Database) and set **Automation account name** as `azadeautomation` (or any other new name). Click **Review + Create**:

Create an Automation Account ...

Basics Advanced Networking Tags Review + Create

Create an Automation Account to hold the Automation runbooks & configuration used for automating operations and management tasks around Azure and non-Azure resources. You could execute cloud jobs in a serverless environment or use hybrid jobs on your compute via Azure Virtual machines or Arc-enabled servers. [Learn more](#)

Subscription * ⓘ Visual Studio Ultimate with MSDN

Resource group * ⓘ packtadesql [Create new](#)

Instance Details

Automation account name * ⓘ azadeautomation

Region * ⓘ East US

Review + Create Previous Next

Figure 5.33 – Automation account creation

5. Open the Azure Automation account and click **Modules**. Click the **Browse gallery** button:

Home > Microsoft.AutomationAccount > azadeautomation

azadeautomation | Modules ⚙ ...

Automation Account

Modules

Add a module Update Az Modules Browse gallery

Shared Resources

Search modules... Module type : All

Name	Status	Type
AuditPolicyDsc	Available	Default

Figure 5.34 – Module installation in an Automation account

6. Search for `sqlServer`. Select the **SqlServer** module. Hit the **Select** button on the next screen:



Figure 5.35 – sqlserver module installation in an Azure Automation account

7. Set **Runtime version** to **7.1** on the **Add a module** screen and hit the **Import** button:

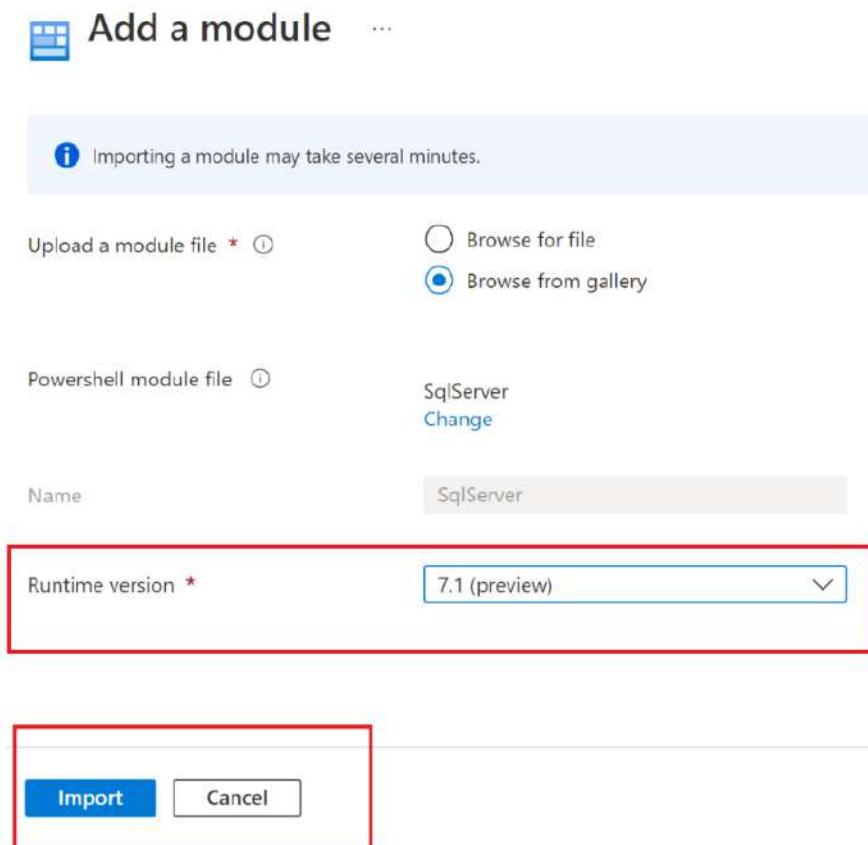


Figure 5.36 – sqlserver module import in an Azure Automation account

8. After adding the module, go to the Azure Automation account **Credentials** in the **Shared Resources** section. We will be providing the SQL credentials required to run the wake-up script. Click **+ Add a credential**:

Home > Microsoft.AutomationAccount > azadeautomation > packtadesql > azadeautomation

The screenshot shows the 'azadeautomation | Credentials' page. At the top, there is a search bar with the placeholder 'credential' and a red box around it. To the right of the search bar is a blue '+' icon labeled 'Add a credential' with a red box around it. Further to the right are 'Refresh' and three dots. Below the search bar, there is a 'Shared Resources' section with a 'Credentials' tab selected, indicated by a red box around it. The main area shows the message 'No credentials found.'

Figure 5.37 – Add a credential to an Azure Automation account

9. Provide the credential name as `sqlcredentials`. Provide the user ID and password (`sqladmin/Sql@Server@1234`) for `azadesqlDb`:

The screenshot shows the 'New Credential' creation form. It has a yellow key icon and the title 'New Credential'. The 'Name *' field contains 'sqlcredentials' with a checkmark. The 'Description' field is empty. The 'User name *' field contains 'sqladmin' with a checkmark. The 'Password *' field contains '*****' with a checkmark. The 'Confirm password *' field also contains '*****' with a checkmark. At the bottom is a blue 'Create' button.

Figure 5.38 – Add a credential to an Azure Automation account

10. After adding the credential, go to the Azure Automation account and then **Runbooks** in the **Process Automation** section. We will be creating the PowerShell script to wake up the database. Click **Create a runbook**:

The screenshot shows the 'azadeautomation | Runbooks' page. On the left, there's a search bar with 'run' and a 'Process Automation' section containing a 'Runbooks' item. On the right, there's a search bar with 'Search runbooks...', a 'Name' filter, and two runbooks listed: 'AzureAutomationTutorialW...' and 'AzureAutomationTutorialW...'. A red box highlights the 'Runbooks' section and the 'Create a runbook' button.

Figure 5.39 – Add a runbook to an Azure Automation account

11. Provide the script name as `sqlwakeupscrip`. Set **Runbook type** to **PowerShell** and **Runtime version** to **7.1 (preview)**, and click the **Create** button:

The screenshot shows the 'Create a runbook' dialog. It has fields for 'Name' (set to 'sqlwakeupscrip'), 'Runbook type' (set to 'PowerShell'), and 'Runtime version' (set to '7.1 (preview)'). Below these, there's a 'Description' field with a placeholder ' '. At the bottom, there's an info message: 'During runbook execution, PowerShell modules targeting 7.1 runtime version will be used. Please make sure the required PowerShell modules are present in 7.1 runtime version.' and two buttons: 'Create' and 'Cancel'.

Figure 5.40 – Create a runbook in an Azure Automation account

12. Copy the following script in the **Edit PowerShell Runbook** window:

```
$SqlCredential = Get-AutomationPSCredential -Name
"sqlcredentials"

# Query to execute
$Query = "select getdate()"

# Execute query
"----- Running SQL Command "
invoke-sqlcmd -ServerInstance "azadesqlserver.
database.windows.
net" -Database "azadesqlldb" -Credential $SqlCredential
-Query "$Query" -Encrypt
``n ----- END SQL Command"
```

Ensure to provide the credential name (sqlcredentials) created in *step 9* for the `-Name` parameter of the `Get-AutomationPSCredential` command. Provide the SQL Server name and database name to be woken up as the values for the `ServerInstance` parameter and the `Database` parameter in the `invoke-sqlcmd` command, as shown in the following screenshot. Hit **Save** and then hit the **Publish** button:

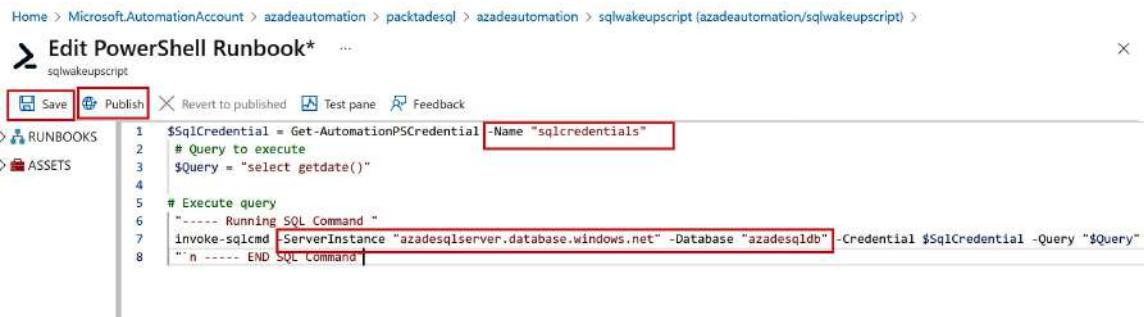


Figure 5.41 – Azure Automation runbook script creation

13. After publishing the script, go to the Azure Automation account, and under the **Resources** section, click **Schedules**. Click on **+ Add a schedule**:

Home > Microsoft.AutomationAccount > azadeautomation > packta

The screenshot shows the 'Runbook' section of the Azure Automation interface. At the top, there's a search bar with 'sche' typed into it, a 'Runbook' button, and a 'Re' button. Below the search bar is a red-bordered 'Add a schedule' button. To its right is a 'Resources' dropdown menu with 'Schedules' selected. On the right side, there's a 'Name' field with 'No schedules found.' below it.

Figure 5.42 – Azure Automation schedule

14. Provide the schedule name and set the **Recurrence** option to **Recurring**. Set the **Recur every** option as **1 Day** and hit the **Create** button:

The screenshot shows the 'New Schedule' creation form. It includes fields for Name (set to 'wakescript-schedule'), Description (empty), Starts (set to 01/08/2022 at 12:56 PM), Time zone (set to 'Singapore - Singapore Standard Time'), Recurrence (set to 'Recurring'), Recur every (set to '1 Day'), Set expiration (set to 'No'), and Expires (set to 'Never'). A red box highlights the 'Recur every' section. At the bottom is a 'Create' button.

Figure 5.43 – The Azure Automation schedule

15. After adding the schedule, go to **All resources** in the Azure portal. Search for the `sqlwakeupscript` runbook. Click **Schedules** under **Resources**. Click **+ Add a schedule**:

Home > azadeautomation > `sqlwakeupscript` (azadeautomation/sqlwakeupscript)

The screenshot shows the Azure portal interface for a runbook named 'sqlwakeupscript'. In the top navigation bar, the URL is 'Home > azadeautomation > sqlwakeupscript (azadeautomation/sqlwakeupscript)'. On the left, there's a sidebar with 'Runbook' and a search bar containing 'sche'. The main area has a title 'sqlwakeupscript (azadeautomation/sqlwakeupscript)' with a 'Runbook' icon. Below it is a 'Resources' section with a 'Schedules' item. To the right, there's a 'Name' field with 'No schedules found.' and a 'Refresh' button. At the top right, there's a '+ Add a schedule' button with a plus sign and a clock icon, which is also highlighted with a red box.

Figure 5.44 – Add a schedule to the runbook

16. Select **Link a schedule to your runbook**:

Home > `sqlwakeupscript` (azadeautomation/sqlwakeupscript) >

The screenshot shows the 'Schedule Runbook' page for 'sqlwakeupscript'. The title is 'Schedule Runbook' with a clock icon and the name 'sqlwakeupscript'. Below it, there's a 'Schedule' section with a 'Link a schedule to your runbook' link, which is highlighted with a red box. There are also links for 'Parameters and run settings' and 'Modify run settings (Default: Azure)'. The entire page is framed by a red border.

Figure 5.45 – Link a schedule to your runbook

17. Click on the **wakescript-schedule** schedule created earlier and click **OK**:

Home > sqlwakeupschedule (azadeautomation/sqlwakeupschedule) > Schedule Runbook >

The screenshot shows the 'Schedules' page for a runbook. At the top, there's a clock icon and the runbook name 'azadeautomation/sqlwakeupschedule'. Below that is a button to 'Add a schedule'. The main table lists one schedule:

Name	Next run
wakescript-schedule	1/9/2022, 12:56 PM

Figure 5.46 – Linking a schedule to a runbook

- Upon creation, the runbook will now have the schedule linked to it, as shown in the following screenshot:

The screenshot shows the 'Runbook' page for 'sqlwakeupschedule'. On the left, there's a sidebar with 'Overview', 'Activity log', 'Tags', and 'Diagnose and solve problems'. The 'Resources' section has 'Jobs' checked and 'Schedules' selected. The main area shows the 'Schedules' section with the 'wakescript-schedule' listed.

Figure 5.47 – Link the schedule to the runbook

- Go to `portal.azure.com`, click on **All resources**, and find the SQL database, **azadesqlDb**. On the **Overview** page, check the status of the database. If it is not paused, close all connections to the database to ensure that it pauses:

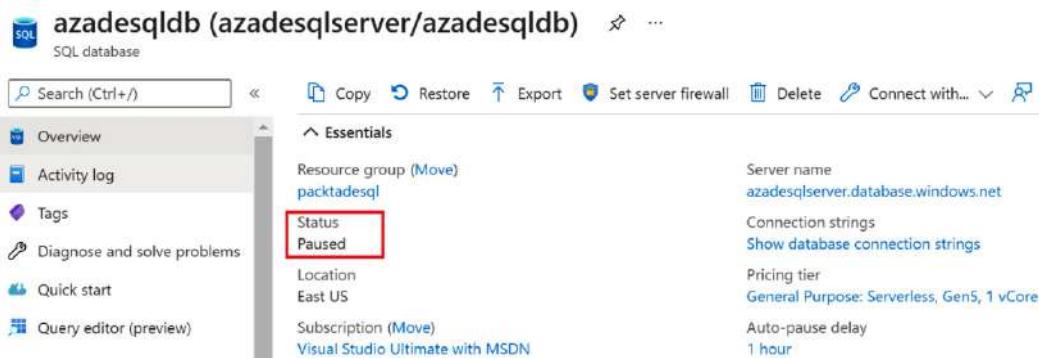


Figure 5.48 – A paused database

20. Go to All resources, search for `sqlwakeupscrip`, and go to the runbook. Hit the Start button:

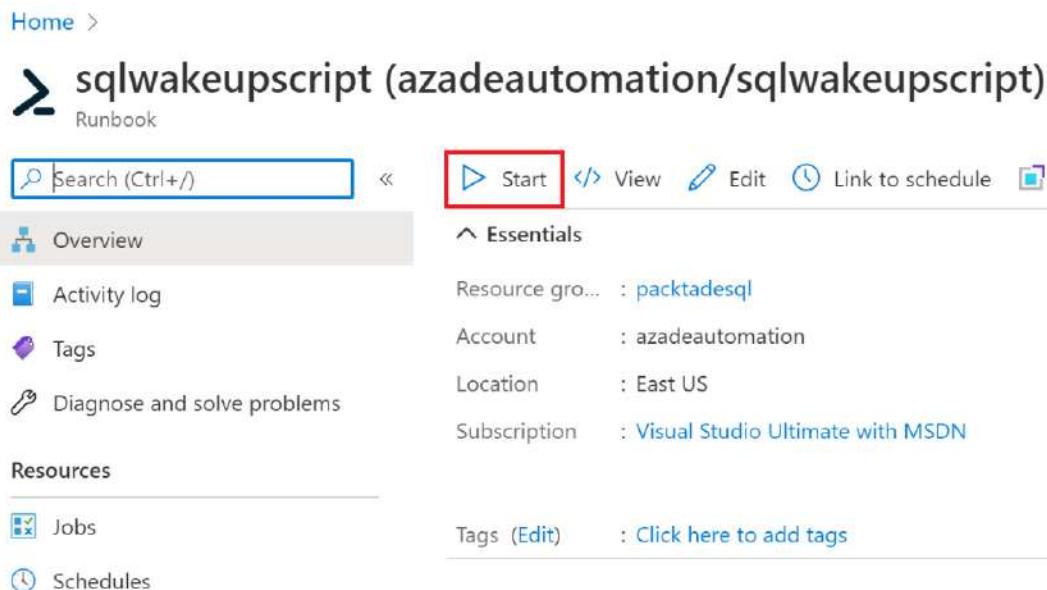


Figure 5.49 – Start the runbook

21. Wait until the job completes. It will take a few minutes. Upon completion, the job status changes to **Completed**, as shown in the following screenshot:

The screenshot shows the Azure Runbook details page for a job named "sqlwakeupscrip". The job status is "Completed". The runbook was triggered by "Azure" and ran by "User". The "Input" tab is selected, showing no parameters were supplied for this job.

Figure 5.50 – The completed runbook

22. Go to `portal.azure.com`, click on **All resources**, and find the SQL database, **azadesqlldb**. On the **Overview** page, check the status of the database. The database status should be either **Resuming** or **Online**, as shown in the following screenshot:

The screenshot shows the Azure portal's "azadesqlldb" resource page. The "Status" is listed as "Online", which is highlighted with a red box. Other details shown include the resource group "packtadesql", location "East US", and subscription "Visual Studio Ultimate with MSDN".

Figure 5.51 – The database status

How it works...

We performed the following steps to configure the serverless tier and provision a wake-up script for the Azure SQL database, **azadesqlDb**:

1. As the first step, we configured **azadesqlDb** on the serverless compute tier, so that it pauses when not in use for 1 hour or longer.
2. As a next step, to configure a wake-up script for **azadesqlDb**, we provisioned an Azure Automation account named **azadeautomation**. An Automation account is required for running any scheduled task in Azure. We can leverage the same automation account for running any other maintenance task too.
3. Next, we created a runbook called `sqlwakeupscrip`t in our **azadeautomation** account. A runbook is used to run any PowerShell script.
4. On the runbook, we added a PowerShell script that connects to the database and runs a simple query against the database. The connection request from the runbook will start the database if it is paused.
5. We added a shared schedule called **wakescript-schedule** to the Automation account and linked it to the `sqlwakeupscrip`t runbook. This schedules the runbook to be executed daily. Having the schedule created in the Automation account as a shared schedule allows us to reuse the schedule if any other job needs to be run at the same frequency.
6. Finally, we started the runbook manually and noticed that the paused database came online.

This recipe is typically useful for UAT and development environments in organizations where users or developers work at certain times during the day. The database can be automatically paused and then resume just before working hours. Pausing the database when not in use and resuming it only when it is needed reduces the cost incurred from Azure SQL Database.

Configuring the Hyperscale tier of Azure SQL Database

Azure SQL Database offers three service tiers – General Purpose, Business Critical, and Hyperscale – when purchased in the vCore model. General Purpose is the most commonly used service tier for medium-sized applications, while the Business Critical tier offers enterprise-class performance and enhanced high-availability options. However, both the General Purpose and Business Critical tiers have a maximum database size limit of 4 TB. A Hyperscale database allows the database to scale up to 100 TB, as well as offering enterprise-class performance and high-availability capabilities. One of the aspects of Hyperscale databases is that we don't need to specify an upper limit for the database size; the database automatically grows as you use it and you pay for the storage you have used.

This recipe will show how to provision the Hyperscale tier of Azure SQL Database.

Getting ready

Log in to portal.azure.com.

How to do it...

Perform the following steps to provision the Hyperscale tier:

1. Go to **Create a resource** in the Azure portal, find **SQL Database**, and click on the **Create** button. On the **Create SQL Database** screen, provide a new resource group if you don't have one already. Provide any database name (a sample one is used in this example). Click **Create new** under **Server** if you don't have a SQL Server created. If you have followed the *Provisioning and connecting to an Azure SQL database using PowerShell* recipe, you would already have a SQL Server named **azadesqlserver**. You can reuse this one:

The screenshot shows the 'Create SQL Database' wizard on the 'Basics' tab. The top navigation bar includes 'Home > Create a resource > Create SQL Database ...'. Below the title, there's a Microsoft logo. The 'Basics' tab is selected, while other tabs like 'Networking', 'Security', 'Additional settings', 'Tags', and 'Review + create' are visible. A note at the top says: '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.' A 'Learn more' link is also present. The 'Project details' section asks to select a subscription and resource group. The 'Subscription' dropdown shows 'Visual Studio Ultimate with MSDN'. The 'Resource group' dropdown shows 'packtadesql' with a red box around it, and a 'Create new' button below it. The 'Database details' section asks for a database name ('sample') and a server ('azadesqlserver (East US)'). The 'Server' dropdown also has a red box around it and a 'Create new' button below it. Both the database name and server dropdowns have checkmarks next to them.

Figure 5.52 – Create a new database

2. Provide new server details, as shown in the following screenshot. Provide the user ID and password as `sqladmin/Sql@Server@1234`:

Server details

Enter required settings for this server, including providing a name and location. This server will be created in the same subscription and resource group as your database.

Server name *	<input type="text" value="azadesqlserver"/> .database.windows.net
Location *	<input type="text" value="(US) East US"/>

Authentication

Select your preferred authentication methods for accessing this server. Create a server admin login and password to access your server with SQL authentication, select only Azure AD authentication [Learn more](#) using an existing Azure AD user, group, or application as Azure AD admin [Learn more](#), or select both SQL and Azure AD authentication.

Authentication method	<input checked="" type="radio"/> Use SQL authentication <input type="radio"/> Use only Azure Active Directory (Azure AD) authentication <input type="radio"/> Use both SQL and Azure AD authentication
Server admin login *	<input type="text" value="sqladmin"/>
Password *	<input type="password" value="*****"/>
Confirm password *	<input type="password" value="*****"/>

Figure 5.53 – Create a new server

3. Click **Configure database**:

[Home > Create a resource >](#)

Create SQL Database

Microsoft

[Basics](#) [Networking](#) [Security](#) [Additional settings](#) [Tags](#) [Review + create](#)

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](#)

Project details

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

Subscription *	<input type="text" value="Visual Studio Ultimate with MSDN"/>
Resource group *	<input type="text" value="packtadesql"/> Create new

Database details

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

Database name *	<input type="text" value="sample"/>
Server *	<input type="text" value="azadesq server (East US)"/> Create new

Want to use SQL elastic pool? * Yes No

Compute + storage *	<input type="text" value="General Purpose"/> Gen5, 2 vCores, 32 GB storage, zone redundant disabled Configure database
---------------------	--

Backup storage redundancy

Figure 5.54 – Configure the Hyperscale tier

- Under Service and compute tier, select Hyperscale (On-demand scalable storage). Check the I understand that scaling from Hyperscale to another service tier is not possible. checkbox and click the Apply button:

Configure ...

 Feedback

Service and compute tier

Select from the available tiers based on the needs of your workload. The vCore model provides a wide range of configuration controls and offers Hyperscale and Serverless to automatically scale your database based on your workload needs. Alternately, the DTU model provides set price/performance packages to choose from for easy configuration. [Learn more](#)

Service tier

Hyperscale (On-demand scalable storage)



[Compare service tiers](#)

Hyperscale tier

In the Hyperscale tier, storage costs are calculated based on actual allocation. Allocated space increases automatically as needed, up to 100 TB.

 The capability to change from Hyperscale to another service tier is not supported. Click here to learn more about this offering and its feature support.

I understand that scaling from Hyperscale to another service tier is not possible.

Compute Hardware

Select the hardware configuration based on your workload requirements. Availability of compute optimized, memory optimized, and confidential computing hardware depends on the region, service tier, and compute tier.

Hardware Configuration

Gen5

up to 80 vCores, up to 408 GB memory

[Change configuration](#)

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



2

High-Availability Secondary Replicas

Increasing the number of High Availability replicas improves availability SLA. [Learn more](#) High Availability replicas can be used for simple read scale scenarios. Consider Named replicas for more complex read scale scenarios. [Learn more](#)



0 Replicas

[Apply](#)

Figure 5.55 – Set the Hyperscale service tier

5. Click **Review + create** on the **Create SQL Database** screen. Hit the **Create** button to create the database:

[Home](#) > [Create a resource](#) >

Create SQL Database

Microsoft

[Basics](#) [Networking](#) [Security](#) [Additional settings](#) [Tags](#) [Review + create](#)

Product details

SQL database
by Microsoft
[Terms of use](#) | [Privacy policy](#)

Estimated cost per month

Compute cost 271.80 USD + Storage cost 0.10 USD / GB

[View pricing details](#)

Terms

By clicking "Create", I (a) agree to the legal terms and privacy statement(s) associated with the Marketplace offering(s) offering(s), with the same billing frequency as my Azure subscription; and (c) agree that Microsoft may share my contact transactional activities. Microsoft does not provide rights for third-party offerings. For additional details see [Azure Mark](#)

Basics

Subscription	Visual Studio Ultimate with MSDN
Resource group	packtadesql
Region	eastus
Database name	sample
Server	azadesqlserver

Compute + storage	Hyperscale: Gen5, 2 vCores
-------------------	----------------------------

Backup storage redundancy	Geo-redundant backup storage
---------------------------	------------------------------

Networking

Allow Azure services and resources to access this server	No
Private endpoint	1 existing

Security

[Create](#)[< Previous](#)[Download a template for automation](#)

Figure 5.56 – Create a Hyperscale database

The preceding steps will create a Hyperscale SQL database that can scale up to 100 TB. As you may have noticed, we didn't specify an upper limit for the database size when creating the database, unlike we would need to with the General Purpose and Business Critical service tiers. The database created would have a starting size of 40 GB, grow if required, and you would only pay for the storage that you consumed.

6

Implementing High Availability and Monitoring in Azure SQL Database

Azure SQL Database, a fundamental relational database as a service offering in Azure acts as a source, destination, or even an intermediate storage layer in data engineering pipelines. Azure SQL Database can be used to consolidate data coming from several relational data sources and build mini data warehouses or data marts. Keeping Azure SQL Database in good health and available at all times is of the utmost importance to any organization. In this chapter, we will explore recipes that will help us monitor the health of a database and ensure database availability. By the end of the chapter, you will have learned how to configure replication, an auto-failover group, auditing, monitoring for Azure SQL Database, and high availability for the Hyperscale tier of Azure SQL Database. This chapter has the following recipes:

- Implementing active geo-replication for an Azure SQL database using PowerShell
- Implementing an auto-failover group for an Azure SQL database using PowerShell
- Configuring high availability to the Hyperscale tier of Azure SQL Database
- Implementing vertical scaling for an Azure SQL database using PowerShell
- Monitoring an Azure SQL database using the Azure portal
- Configuring auditing for Azure SQL Database

Implementing active geo-replication for an Azure SQL database using PowerShell

The active geo-replication feature allows you to create up to four readable databases for a primary Azure SQL Database. Active geo-replication uses **SQL Server AlwaysOn** to asynchronously replicate transactions to the secondary databases. The secondary database can be in the same or a different region from the primary database.

Active geo-replication can be used for the following cases:

- To provide business continuity by failing over to the secondary database in case of a disaster. The failover is manual.
- To offload reads to the readable secondary database.
- To migrate a database to a different server in another region.

In this recipe, we'll configure active geo-replication for an Azure SQL database and perform a manual failover.

Getting ready

In a new PowerShell window, execute the `Connect-AzAccount` command and follow the steps to log in to your Azure account.

You need an existing Azure SQL database for this recipe. If you don't have one, create an Azure SQL database by following the steps mentioned in the *Provisioning and connecting to an Azure SQL database using PowerShell* recipe in *Chapter 5, Configuring and Securing Azure SQL Database*.

How to do it...

First, let's create a readable secondary database.

Creating a readable secondary

The steps are as follows:

1. Execute the following command to provision a new Azure SQL Server to host the secondary replica:

```
#create credential object for the Azure SQL Server admin
credential
$sqladminpassword = ConvertTo-SecureString 'Sql@
Server@1234' -AsPlainText -Force
$sqladmincredential = New-Object System.Management.
```

```
Automation.PSCredential ('sqladmin', $sqladminpassword)
New-AzSqlServer -ServerName azadesqlsecondary
-SqlAdministratorCredentials $sqladmincredential
-Location westus -ResourceGroupName packtadesql
```

2. Execute the following command to configure the geo-replication from the primary server to the secondary server:

```
$primarydb = Get-AzSqlDatabase -DatabaseName azadesqldb
-ServerName azadesqlserver -ResourceGroupName packtadesql
$primarydb | New-AzSqlDatabaseSecondary
-PartnerResourceGroupName packtadesql -PartnerServerName
azadesqlsecondary -AllowConnections "All"
```

You should get an output as shown in the following screenshot:

The screenshot shows a PowerShell window with two command lines. The first command retrieves the primary database object, and the second command configures it as a secondary database with specific parameters. The output at the bottom shows the properties of the newly configured secondary database.

```
PS C:\Users\naveenkat> $primarydb = Get-AzSqlDatabase -DatabaseName azadesqldb -ServerName azadesqlserver -ResourceGroupName packtadesql
WARNING: Upcoming breaking changes in the cmdlet 'Get-AzSqlDatabase' :
- The output type 'Microsoft.Azure.Commands.Sql.Database.Model.AzureSqlDatabaseModel' is changing
- The following properties in the output type are being deprecated : 'BackupStorageRedundancy'
- The following properties are being added to the output type : 'CurrentBackupStorageRedundancy' 'RequestedBackupStorageRedundancy'
- The change is expected to take effect from the version : '3.0.0'
Note : Go to https://aka.ms/azps-changewarnings for steps to suppress this breaking change warning, and other information on breaking changes in Azure PowerShell.

PS C:\Users\naveenkat> $primarydb | New-AzSqlDatabaseSecondary -PartnerResourceGroupName packtadesql -PartnerServerName azadesqlsecondary -AllowConnections "All"
WARNING: Upcoming breaking changes in the cmdlet 'New-AzSqlDatabaseSecondary' :
- The output type 'Microsoft.Azure.Commands.Sql.Replication.Model.AzureReplicationLinkModel' is changing
- The following properties in the output type are being deprecated : 'BackupStorageRedundancy'
- The following properties are being added to the output type : 'CurrentBackupStorageRedundancy' 'RequestedBackupStorageRedundancy'
- The change is expected to take effect from the version : '3.0.0'
Note : Go to https://aka.ms/azps-changewarnings for steps to suppress this breaking change warning, and other information on breaking changes in Azure PowerShell.

LinkId          : c6767132-c80a-4c3d-81d4-6cc8a396a8c8
ResourceGroupName : packtadesql
ServerName       : azadesqlserver
DatabaseName     : azadesqldb
Role             : Primary
Location         : East US
PartnerResourceGroupName : packtadesql
PartnerServerName : azadesqlsecondary
PartnerDatabaseName : azadesqldb
PartnerRole       : Secondary
PartnerLocation   : West US
AllowConnections  : All
ReplicationState  : CATCH_UP
PercentComplete   : 100
StartTime        : 16/1/2022 11:43:12 am
```

Figure 6.1 – Configuring geo-replication

Moreover, we can also check geo-replication configuration on the Azure portal, as shown in the following screenshot. Go to the **azadesqldb** database and, under **Data management**, click **Replicas**:

The screenshot shows the Azure portal interface for managing databases. The left sidebar has a tree view with 'azadesqlldb (azadesqlserver/azadesqlldb)' selected. The main content area shows the database details with a red box around the 'Replicas' section. The table lists the following data:

Name	Server	Region	Failover policy	Pricing tier	Replica state
azadesqlldb	azadesqlserver	East US	None	Basic	Online
azadesqlldb	azadesqlsecondary	West US		Basic	Readable

Figure 6.2 – Verifying geo-replication in the Azure portal

Performing manual failover to the secondary

The steps are as follows:

1. Execute the following command to manually fail over to the secondary database:

```
$secondarydb = Get-AzSqlDatabase -DatabaseName azadesqlldb
-ServerName azadesqlsecondary -ResourceGroupName
packtadesql
$secondarydb | Set-AzSqlDatabaseSecondary
-PartnerResourceGroupName packtadesql -Failover
```

The preceding command performs a planned failover without data loss. To perform a manual failover with data loss, use the `Allowdataloss` switch.

If we check the Azure portal, we'll see that **azadesqlsecondary/azadesqlldb** in **West US** is the primary database:

Home > packtadesql > azadesql (azadesqlserver/azadesql) > azadesql (azadesqlserver/azadesql)

Name ↑↓	Server ↑↓	Region ↑↓	Failover policy ↑↓	Pricing tier ↑↓
azadesql	azadesqlsecondary	West US	None	Basic
azadesql	azadesqlserver	East US		Basic

Figure 6.3 – Failing over to the secondary server

2. We can also get the active geo-replication information by executing the following command:

```
$Get-AzSqlDatabaseReplicationLink -DatabaseName
azadesql -PartnerResourceGroupName packtadesql
-PartnerServerName azadesqlsecondary -ServerName
azadesqlserver -ResourceGroupName packtadesql
```

You should get an output as shown in the following screenshot:

```
PS C:\Users\naveenkat> $Get-AzSqlDatabaseReplicationLink -DatabaseName azadesql -PartnerResourceGroupName packtadesql -PartnerServerName azadesqlsecondary -ServerName azadesqlserver -ResourceGroupName packtadesql
WARNING: Upcoming breaking changes in the cmdlet 'Get-AzSqlDatabaseReplicationLink':
- The output type 'Microsoft.Azure.Commands.Sql.Replication.Model.AzureReplicationLinkModel' is changing
- The following properties in the output type are being deprecated : 'BackupStorageRedundancy'
- The following properties are being added to the output type : 'CurrentBackupStorageRedundancy' 'RequestedBackupStorageRedundancy'
- The change is expected to take effect from the version : '3.0.0'
Note : Go to https://aka.ms/azps-changewarnings for steps to suppress this breaking change warning, and other information on breaking changes in Azure PowerShell.

LinkId          : c67e7132-c88a-4c3d-81d4-6cc8a396a8c8
ResourceGroupName : packtadesql
ServerName       : azadesqlserver
DatabaseName     : azadesql
Role             : Secondary
Location         : East US
PartnerResourceGroupName : packtadesql
PartnerServerName : azadesqlsecondary
PartnerDatabaseName : azadesql
PartnerRole      : Primary
PartnerLocation   : West US
AllowConnections  : All
ReplicationState : CATCH_UP
PercentComplete   : 100
StartTime        : 16/1/2022 11:43:12 am

PS C:\Users\naveenkat>
```

Figure 6.4 – Getting the geo-replication status

Removing active geo-replication

Execute the following command to remove the active geo-replication link between the primary and secondary databases:

```
$primarydb = Get-AzSqlDatabase -DatabaseName azadesqldb
-ServerName azadesqlserver -ResourceGroupName packtadesql
$primarydb | Remove-AzSqlDatabaseSecondary
-PartnerResourceGroupName packtadesql -PartnerServerName
azadesqlsecondary
```

You should get an output as shown in the following screenshot:

```
PS C:\Users\navenvkat> $primarydb = Get-AzSqlDatabase -DatabaseName azadesqldb -ServerName azadesqlserver -ResourceGroupName packtadesql
WARNING! Upcoming breaking changes in the cmdlet 'Get-AzSqlDatabase':
- The output type 'Microsoft.Azure.Commands.Sql.Database.Model.AzureSqlDatabaseModel' is changing
- The following properties in the output type are being deprecated : 'BackupStorageRedundancy'
- The following properties are being added to the output type : 'CurrentBackupStorageRedundancy' 'RequestedBackupStorageRedundancy'
- The change is expected to take effect from the version : '3.0.0'
Note : Go to https://aka.ms/azps-changewarnings for steps to suppress this breaking change warning, and other information on breaking changes in Azure PowerShell.
PS C:\Users\navenvkat> $primarydb | Remove-AzSqlDatabaseSecondary -PartnerResourceGroupName packtadesql -PartnerServerName azadesqlsecondary
WARNING! Upcoming breaking changes in the cmdlet 'Remove-AzSqlDatabaseSecondary':
- The output type 'Microsoft.Azure.Commands.Sql.Replication.Model.AzureReplicationLinkModel' is changing
- The following properties in the output type are being deprecated : 'BackupStorageRedundancy'
- The following properties are being added to the output type : 'CurrentBackupStorageRedundancy' 'RequestedBackupStorageRedundancy'
- The change is expected to take effect from the version : '3.0.0'
Note : Go to https://aka.ms/azps-changewarnings for steps to suppress this breaking change warning, and other information on breaking changes in Azure PowerShell.

LinkID          : c6767132-c80a-4c3d-81d4-6cc8a396a8c8
ResourceGroupName : packtadesql
ServerName       : azadesqlserver
DatabaseName     : azadesqldb
Role             : Secondary
Location         : East US
PartnerResourceGroupName : packtadesql
PartnerServerName  : azadesqlsecondary
PartnerDatabaseName : azadesqldb
PartnerRole       : Primary
PartnerLocation    : West US
AllowConnections   : All
ReplicationState  : CATCH_UP
PercentComplete    : 100
StartTime         : 16/1/2022 11:43:12 am
```

Figure 6.5 – Removing active geo-replication

How it works...

To configure active geo-replication, we use the `New-AzSqlDatabaseSecondary` command. This command expects the primary database name, server name, and resource group name; the secondary resource group name and server name; and the **Allow connections** parameter. If we want a readable secondary, then we set **Allow connections** to **All**; otherwise, we set it to **No**.

The active geo-replication provides manual failover with and without data loss. To perform a manual failover, we use the `Set-AzSqlDatabaseSecondary` command. This command expects the secondary server name, the database name, the resource group name, a failover switch, and the `Allowdataloss` switch in case of failover with data loss.

To remove active geo-replication, we use the `Remove-AzSqlDatabaseSecondary` command. This command expects the secondary server name, secondary database name, and resource name to remove the replication link between the primary and secondary databases.

Removing active geo-replication doesn't remove the secondary database.

Implementing an auto-failover group for an Azure SQL database using PowerShell

An auto-failover group allows a group of databases to fail to a secondary server in another region if the SQL database service in the primary region fails. Unlike active geo-replication, the secondary server should be in a different region from the primary. The secondary databases can be used to offload read workloads. The failover can be manual or automatic.

In this recipe, we'll create an auto-failover group, add databases to the auto-failover group, and perform a failover to the secondary server.

Getting ready

In a new PowerShell window, execute the `Connect-AzAccount` command and follow the steps to log in to your Azure account.

You will need an existing Azure SQL database for this recipe. If you don't have one, create an Azure SQL database by following the steps mentioned in the *Provisioning and connecting to an Azure SQL database using PowerShell* recipe of *Chapter 5, Configuring and Securing Azure SQL Database*.

How to do it...

First, let's create an auto-failover group.

Creating an auto-failover group

The steps are as follows:

1. Execute the following PowerShell command to create a secondary server. The server should be in a different region than the primary server:

```
#create credential object for the Azure SQL Server admin
credential
$sqladminpassword = ConvertTo-SecureString 'Sql@
Server@1234' -AsPlainText -Force
$sqladmincredential = New-Object System.Management.
Automation.PSCredential ('sqladmin', $sqladminpassword)
```

```
New-AzSqlServer -ServerName azadesqlsecondary
-SqlAdministratorCredentials $sqladmincredential
-Location westus -ResourceGroupName packtadesql
```

2. Execute the following command to create the auto-failover group:

```
New-AzSqlDatabaseFailoverGroup -ServerName azadesqlserver
-FailoverGroupName adefg -PartnerResourceGroupName
packtadesql -PartnerServerName azadesqlsecondary
-FailoverPolicy Automatic -ResourceGroupName packtadesql
```

You should get an output as shown in the following screenshot:

```
PS C:\Users\naveenkat> New-AzSqlDatabaseFailoverGroup -ServerName azadesqlserver -FailoverGroupName adefg -PartnerResourceGroupName packtadesql
-PartnerServerName azadesqlsecondary -FailoverPolicy Automatic -ResourceGroupName packtadesql

FailoverGroupName          : adefg
Location                  : East US
ResourceGroupName          : packtadesql
ServerName                : azadesqlserver
PartnerLocation           : West US
PartnerResourceGroupName   : packtadesql
PartnerServerName          : azadesqlsecondary
ReplicationRole            : Primary
ReplicationState           : CATCH_UP
ReadWriteFailoverPolicy    : Automatic
FailoverWithDataLossGracePeriodHours : 1
DatabaseNames              : {}
```

Figure 6.6 – Creating an auto-failover group

3. Execute the following command to add an existing database to the auto-failover group:

```
$db = Get-AzSqlDatabase -DatabaseName azadesqlldb
-ServerName azadesqlserver -ResourceGroupName packtadesql
$db | Add-AzSqlDatabaseToFailoverGroup -ResourceGroupName
packtadesql -ServerName azadesqlserver -FailoverGroupName
adefg
```

4. Execute the following command to add a new Azure SQL database to the auto-failover group:

```
$db = New-AzSqlDatabase -DatabaseName azadesqlldb2
-Edition basic -ServerName azadesqlserver
-ResourceGroupName packtadesql
$db | Add-AzSqlDatabaseToFailoverGroup -FailoverGroupName
adefg
```

5. Execute the following PowerShell command to get the details about the auto-failover group:

```
Get-AzSqlDatabaseFailoverGroup -ServerName azadesqlserver
-FailoverGroupName adefg -ResourceGroupName packtadesql
```

You should get an output as shown in the following screenshot:

```
PS C:\Users\naveenk> Get-AzSqlDatabaseFailoverGroup -ServerName azadesqlserver -FailoverGroupName adefg -ResourceGroupName packtadesql

FailoverGroupName          : adefg
Location                  : East US
ResourceGroupName          : packtadesql
ServerName                : azadesqlserver
PartnerLocation           : West US
PartnerResourceGroupName   : packtadesql
PartnerServerName          : azadesqlsecondary
ReplicationRole            : Primary
ReplicationState           : CATCH_UP
ReadWriteFailoverPolicy    : Automatic
FailoverWithDataLossGracePeriodHours : 1
DatabaseNames              : {azadesqldb, azadesqldb2}
```

Figure 6.7 – Getting the auto-failover group details

The endpoint used to connect to the primary server of an auto-failover group is in the <auto-failover group name>.database.windows.net form. In our case, this will be adefg.database.windows.net.

To connect to a readable secondary in an auto-failover group, the endpoint used is in the <auto-failover group name>.secondary.database.windows.net form. In our case, the endpoint will be adefg.secondary.database.windows.net. In addition to this, we need to specify **ApplicationIntent** as **readonly** in the connection string when connecting to the secondary.

6. In the Azure portal, the failover groups can be found on the Azure SQL Server page, as shown in the following screenshot. Go to **azadesqlserver** and, under **Data management**, click on **Failover groups**:

Name	Primary server	Secondary server	Read/Write failover policy	Grace Period (minutes)	Database count
adefg	azadesqlserver	azadesqlsecondary	Automatic	60	2/2

Figure 6.8 – Viewing an auto-failover group in the Azure portal

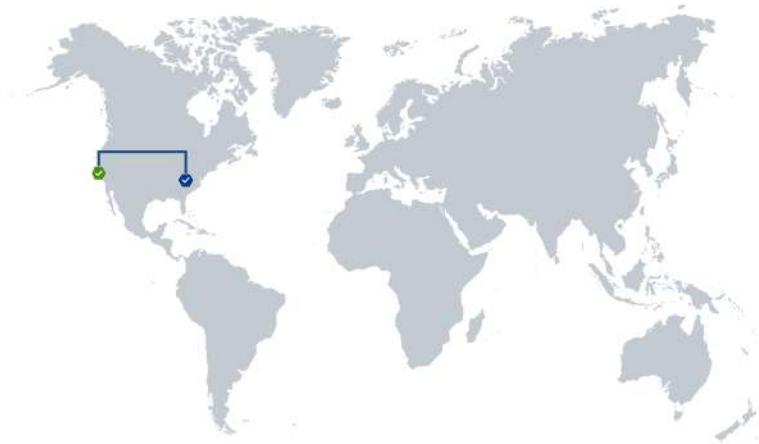
7. To open the failover group details, click on the failover group name, **adefg**:

[Home](#) > [SQL databases](#) > [azadesqlldb \(azadesqlserver/azadesqlldb\)](#) > [azadesqlserver](#) >

adefg ...
azadesqlserver

 Save  Discard  Add databases  Edit configuration  Remove databases  Failover  Forced Failover  Delete

Configuration details Databases within group Databases selected to be added (0) Databases selected for removal (0)



Server	Role	Read/Write failover policy	Grace period
 azadesqlserver (East US)	Primary	Automatic	1 hours
 azadesqlsecondary (West US)	Secondary		

Figure 6.9 – Viewing the auto-failover group details in the Azure portal

Performing a failover to the secondary server

The steps are as follows:

1. Execute the following command to fail over to the secondary server:

```
$secondarysqlserver = Get-AzSqlServer -ResourceGroupName packtadesql -ServerName azadesqlsecondary  
$secondarysqlserver | Switch-AzSqlDatabaseFailoverGroup -FailoverGroupName addefg
```

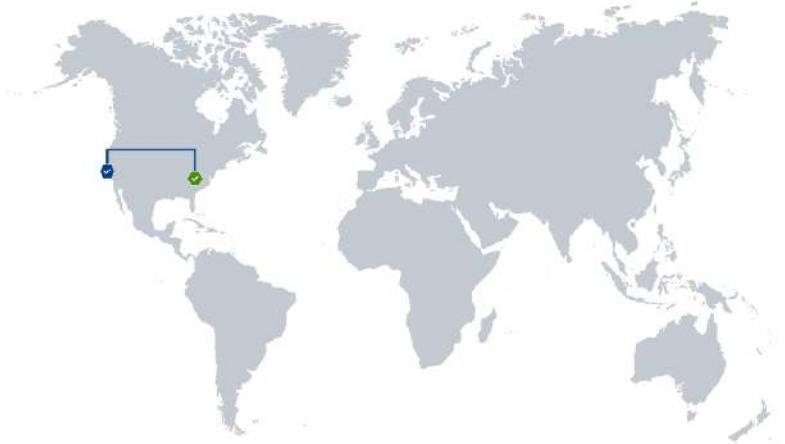
If we check in the Azure portal, the primary server is now **azadesqlsecondary** and the secondary server is **azadesqlserver**, as shown in the following screenshot:

Home > SQL databases > azadesql (azadesqlserver/azadesql) > azadesqlserver >

adefg ...
azadesqlserver

Save Discard Add databases Edit configuration Remove databases Failover Forced Failover Delete

Configuration details Databases within group Databases selected to be added (0) Databases selected for removal (0)



Server	Role	Read/Write failover policy	Grace period
azadesqlsecondary (West US)	Primary	Automatic	1 hours
azadesqlserver (East US)	Secondary		

Figure 6.10 – The failover to the secondary server

2. Execute the following command to remove the auto-failover group. Removing the auto-failover group doesn't remove the secondary or primary SQL databases:

```
Remove-AzSqlDatabaseFailoverGroup -ServerName  
azadesqlsecondary -FailoverGroupName adfg  
-ResourceGroupName packtadesql
```

You should get an output as shown in the following screenshot:

```
PS C:\Users\navenkat> Remove-AzSqlDatabaseFailoverGroup -ServerName azadesqlsecondary -FailoverGroupName adefg -ResourceGroupName packtadessql

FailoverGroupName          : adefg
Location                  : West US
ResourceGroupName          : packtadessql
ServerName                : azadesqlsecondary
PartnerLocation           : East US
PartnerResourceGroupName   : packtadessql
PartnerServerName          : azadesqlserver
ReplicationRole            : Primary
ReplicationState           : CATCH_UP
ReadWriteFailoverPolicy    : Automatic
FailoverWithDataLossGracePeriodHours : 1
DatabaseNames              : {azadesqldb, azadesqldb2}

PS C:\Users\navenkat>
```

Figure 6.11 – Removing the auto-failover group

How it works...

The `New-AzSqlDatabaseFailoverGroup` command is used to create an auto-failover group. We need to specify the auto-failover group name, the primary and secondary server names, the resource group name, and the failover policy (either automatic or manual). In addition to this, we can also specify `GracePeriodWithDataLossHours`. As the replication between the primary and secondary is asynchronous, the failover may result in data loss.

The `GracePeriodWithDataLossHours` value specifies how many hours the system should wait before initiating the automatic failover. This can therefore limit the data loss that happens because of a failover.

After the auto-failover group creation, we can add the databases to the auto-failover group by using the `Add-AzSqlDatabaseToFailoverGroup` command. The database to be added should exist on the primary server and not on the secondary server.

We can perform a failover by executing the `Switch-AzSqlDatabaseFailoverGroup` command. We need to provide the primary server's name, the auto-failover group name, and the primary server resource group name.

To remove an auto-failover group, execute the `Remove-AzSqlDatabaseFailoverGroup` command by specifying the primary server's name and resource group, and the auto-failover group name.

Configuring high availability to the Hyperscale tier of Azure SQL Database

Azure SQL Database in the Hyperscale tier supports databases with up to 100 TB of storage size, compared to the 4 TB offered in the General Purpose and Business Critical tiers. In addition to supporting large databases, the Hyperscale tier offers high-availability capabilities too, with the three following types of replica:

- **Geo-replica:** Allows the creation of asynchronous replicas in the same or a different region, allowing users to have a read-only copy of the database
- **High-availability replica:** Acts as a hot standby server, offering automatic failover
- **Named replica:** Allows the creation of replicas with a different database name from the primary database, mainly used for supporting read scale-out scenarios

Both high-availability and named replicas are configured on synchronous mode, and can be created in the same location as a primary database.

In this recipe, we will configure a named replica for a Hyperscale database and execute read-only queries against the secondary database. We will also showcase how a secondary replica will continue to function even when a primary replica is under maintenance due to a scale-up activity.

Getting ready

Create a Hyperscale Azure SQL database, as mentioned in the *Configuring the Hyperscale tier of Azure SQL Database* recipe of Chapter 5, *Configuring and Securing Azure SQL Database*.

Download and install **SQL Server Management Studio (SSMS)** on your local machine. SSMS can be downloaded from <https://docs.microsoft.com/en-us/sql/ssms/download-sql-server-management-studio-ssms?view=sql-server-ver15>.

How to do it...

Follow these steps to create a secondary replica for a Hyperscale database:

1. Log in to `portal.azure.com`, click **All resources**, and find **azadesqlserver**. Go to the **HyperScale Azure SQL database** created. Under **Data management**, click **Replicas** and then click **Create replica**:

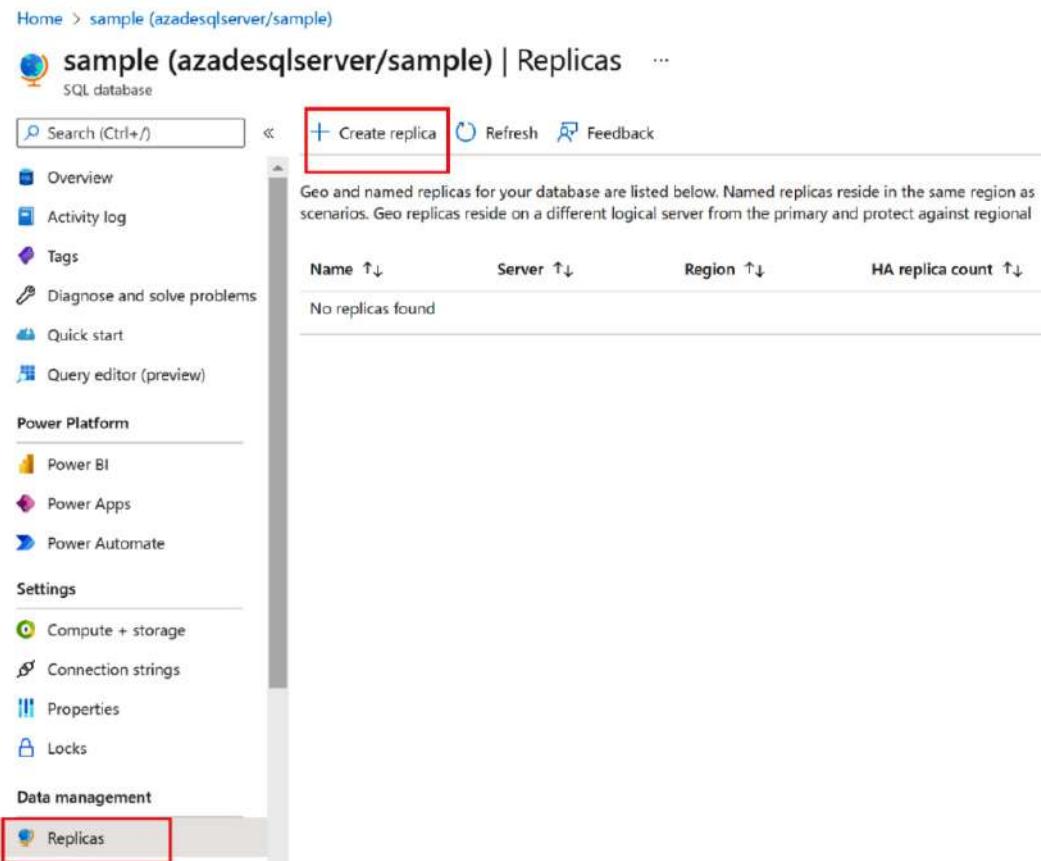


Figure 6.12 – Adding a replica

2. Select **Named replica** for **Replica type**. **Named replica** allows you to create another database that can be used as a read-only replica in the same region as the primary. Our primary database is named **sample** and our secondary is named **sample_NamedReplica**. You may choose to create a new logical SQL server if you wish but, in this example, we will reuse the same logical server. Hit the **Review + create** button:

Home > azadesqlserver > sample (azadesqlserver/sample) >

Create SQL Database - Replica

Microsoft

Primary database details

Additional settings will be defaulted where possible based on the primary database.

Primary database

sample

Region

eastus

Replica configuration

Choose a replica type. Geo and named replicas both offer independent compute + storage and security configuration from the primary, as well as an accessible endpoint. [Learn more](#)

Replica type *

- Geo replica - Resides on a different logical server from the primary, protects against prolonged region outages.
- Named replica - Resides in the same region as the primary, enables offloading of read-only workloads.

Database details

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

Database name *

sample_NamedReplica

Server * ⓘ

azadesqlserver (East US)

[Create new](#)

Region

East US

Want to use SQL elastic pool? ⓘ

Yes No

Compute + storage * ⓘ

Hyperscale

Gen5, 2 vCores

[Configure database](#)

[Review + create](#)

[Next : Review + create >](#)

Figure 6.13 – Configuring the replica

3. Once the replica has been created, go to **azadesqlserver** in [portal.azure.com](#) and click **Firewalls and virtual networks**. Click **Add client IP**. After the IP is listed as a rule, click on the **Save** button. We connect to the primary and secondary database from a local client machine, and hence the local machine IP needs to be whitelisted:

The screenshot shows the 'Firewalls and virtual networks' settings for the 'azadesqlserver' SQL server. At the top, there is a search bar with 'Fire' and three buttons: 'Save' (highlighted with a red box), 'Discard', and '+ Add client IP' (also highlighted with a red box). Below this, there are two sections: 'Settings' and 'Security'. Under 'Settings', there is an 'Azure Active Directory' section with a checkbox for 'Deny public network access' which is unchecked. Under 'Security', there is a 'Firewalls and virtual networks' section (highlighted with a red box) containing a 'Minimum TLS Version' dropdown set to '1.2', a 'Connection Policy' section with 'Default' selected, and a checkbox for 'Allow Azure services and resources to access this server' which is checked ('Yes'). Below these, the 'Client IP address' is listed as '116.89.64.165'. A table below shows a single rule entry: 'ClientIPAddress_2022-1...' with 'Start IP' '116.89.64.165' and 'End IP' '116.89.64.165'.

Rule name	Start IP	End IP
ClientIPAddress_2022-1...	116.89.64.165	116.89.64.165

Figure 6.14 – Firewalls and virtual networks

4. Connect to **azadesqlserver.database.windows.net** using SSMS with the user ID and the password set to `sqladmin/Sql@Server@1234`. Right-click on the **sample_NamedReplica** database and click **New Query** to open a connection to it:

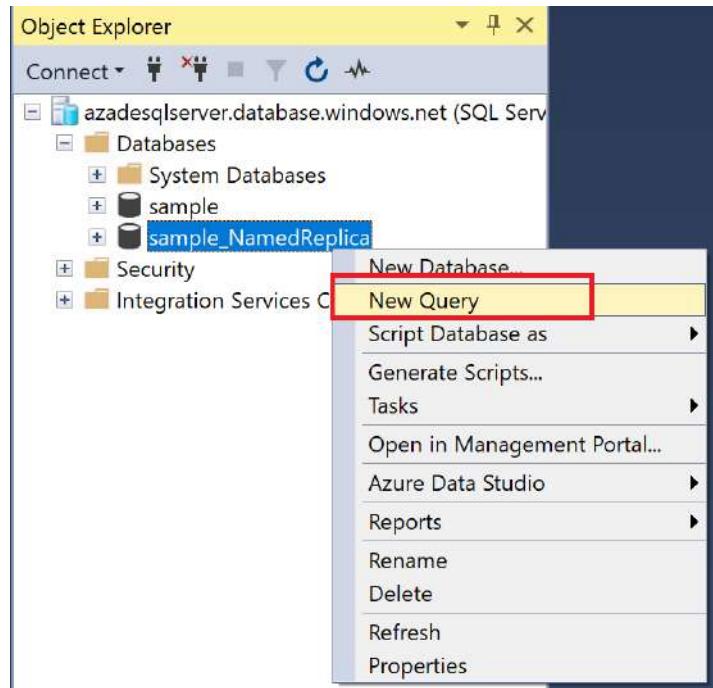


Figure 6.15 – New Query

5. Execute the following command:

```
Select db_name() as database_name  
GO
```

Notice that the query executes successfully on **sample_NamedReplica**, as it just performs a read-only operation:

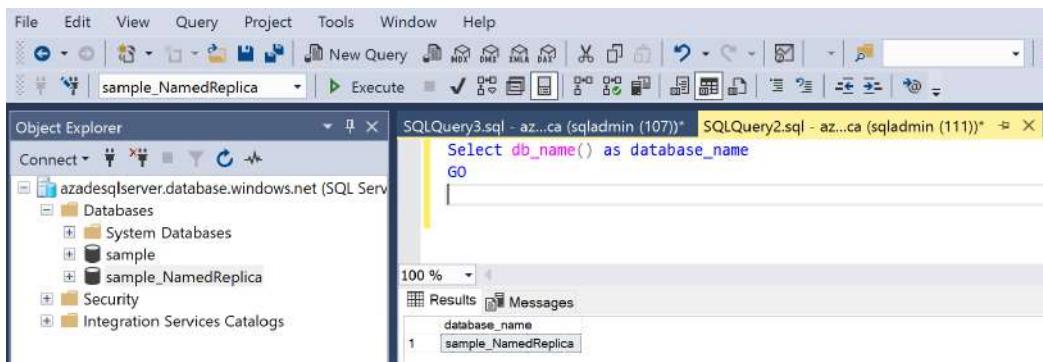


Figure 6.16 – A read-only query

6. Notice that when you perform a read-write operation such as table creation on **sample_NamedReplica**, it fails:

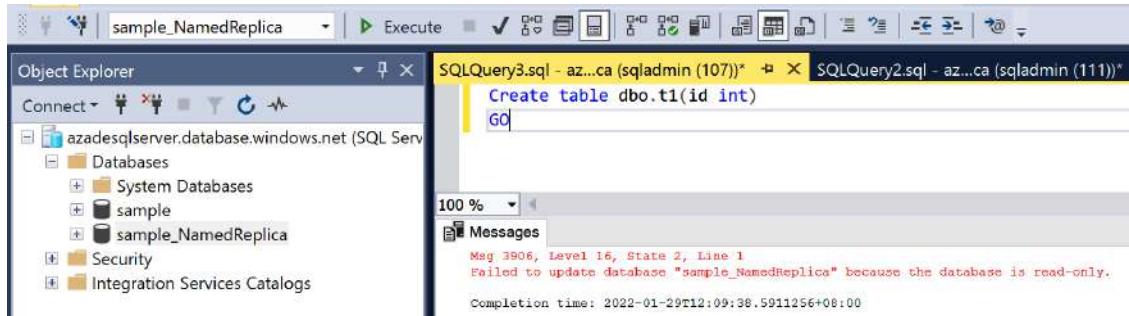


Figure 6.17 – A read-write failure

7. Let's perform the following experiment. We will perform a scale-up operation on the primary database (**sample**), and while the scale-up is in progress, we will perform read-only queries against the secondary database. Due to Azure SQL Database's high-availability capabilities, there will be no connection drop in the secondary database when the primary database is scaled up.

Execute the following script against the secondary database (**sample_NamedReplica**). The script will continuously run read-only queries against the secondary database at an interval of 1 second:

```
While 1=1
Begin
Select * from sys.objects;
WAITFOR DELAY '0:00:01'
End
```

The following is a screenshot of the script execution. As expected, the script reads the secondary database continuously:

The screenshot shows the Object Explorer pane on the left with a connection to 'azadesqlserver.database.windows.net (SQL Server)'. Under 'sample_NamedReplica', there are 'Databases', 'System Databases', 'sample', and 'Security'. The 'Security' node is expanded, showing 'sa' and 'sa (sqladmin)'. The 'Messages' pane at the bottom right shows the continuous execution of the script: 'While 1=1' and 'Select * from sys.objects'. The 'Results' pane shows the output of the 'sys.objects' query, listing various system objects like syscolumns, sysconstraints, sysindexes, etc., across multiple pages.

Figure 6.18 – A continuous read-only query

8. Go to portal.azure.com, find **azadesqlserver**, and go to the sample database. Click **Compute + storage**. Under **vCores**, scale up from 2 vCores to 8 and hit the **Apply** button:

The screenshot shows the Azure portal interface for managing a SQL database named 'sample (azadesqlserver/sample)'. The left sidebar contains navigation links for Overview, Activity log, Tags, Diagnose and solve problems, Quick start, Query editor (preview), Power Platform (Power BI, Power Apps, Power Automate), Settings (Compute + storage, Connection strings, Properties, Locks), Data management (Replicas), Integrations (Stream analytics (preview), Add Azure Search), and Security (Auditing, Ledger). The 'Compute + storage' link is highlighted with a red box.

In the main content area, the 'Service tier' dropdown is set to 'Hyperscale (On-demand scalable storage)'. A note states: 'In the Hyperscale tier, storage costs are calculated based on actual allocation. Allocated space increases automatically as needed, up to 100 TB.' A warning message indicates: 'The capability to change from Hyperscale to another service tier is not supported. Click here to learn more about this offering and its feature support.' Below this, a checkbox is checked with the message: 'I understand that scaling from Hyperscale to another service tier is not possible.'

The 'Compute Hardware' section allows selecting hardware configuration based on workload requirements. It shows 'Gen5' hardware with 'up to 80 vCores, up to 408 GB memory' and a 'Change configuration' link. The 'vCores' slider is highlighted with a red box, showing it is currently set to 2. A note says: 'It's strongly recommended that the primary and secondary databases have the same compute size. Learn more'.

The 'High-Availability Secondary Replicas' section shows a slider for the number of replicas, currently set to 0. A note says: 'Increasing the number of High Availability replicas improves availability SLA. High Availability replicas can be used for simple read scale scenarios. Consider Named replicas for more complex read scale scenarios. Learn more'.

At the bottom, a large blue 'Apply' button is highlighted with a red box.

Figure 6.19 – Scaling up the primary database

9. While scale-up is in progress, go back to SSMS and check whether the query is running without errors against the secondary database (**sample_NamedReplica**). You will notice that the query was running just fine when the primary was being scaled up:

The screenshot shows the Object Explorer on the left with the database 'azadesqserver.database.windows.net (SQL Server)' selected. In the center, there are two tabs: 'SQLQuery3.sql - az...ca (sqladmin (107))' and 'SQLQuery2.sql - az...a (111) Executing...'. The second tab is active, displaying the following T-SQL code:

```


DECLARE @i INT = 1;
WHILE @i <= 10
BEGIN
    SELECT * FROM sys.objects
    WAITFOR DELAY '0:00:01'
    SET @i = @i + 1
END


```

Below the code, the 'Messages' pane shows several rows of output from the query. At the bottom of the pane, a red box highlights the status bar which reads 'Executing query...'.

The results pane displays the output of the query, showing 10 rows of data from the 'sys.objects' system table. The columns include name, object_id, principal_id, schema_id, parent_object_id, type, type_desc, create_date, modify_date, is_ms_shipped, is_published, and is_schema_published. The results show that the query ran successfully on the secondary database.

Figure 6.20 – Read-only query with no errors

How it works...

For any operation to be performed in a database, we need two key components. They are the compute, which is responsible for processing the query, and the storage, which actually stores the data. While creating replicas for a Hyperscale database, Azure doesn't copy any data or provision any additional storage. Data stored is common for both primary and secondary replicas. For a secondary replica, a logical database is created, and additional vCores are provisioned to handle compute operations required for read-only queries. Even though the database name of a read-only replica is different, it will query the same data files and facilitate read-only workloads. As the compute and storage are decoupled components in Azure SQL Database, there is no disruption to the connections of the secondary replica, even when the compute is scaled up (or scaled down) in the primary database.

Having a secondary replica offers read scale-out, excellent resilience, and high availability to mission-critical applications.

Implementing vertical scaling for an Azure SQL database using PowerShell

An Azure SQL Database has multiple purchase models and service tiers for different workloads. There are two purchasing models: **DTU-based** and **vCore-based**. There are multiple service tiers within these purchasing models.

Having multiple service tiers provides us with the flexibility to scale up or scale down based on the workload or activity in an Azure SQL database.

In this recipe, we'll learn how to automatically scale up an Azure SQL Database whenever the CPU percentage is above 40%.

Getting ready

In a new PowerShell window, execute the `Connect-AzAccount` command and follow the steps to log in to your Azure account.

You will need an existing Azure SQL database for this recipe. If you don't have one, create an Azure SQL database by following the steps mentioned in the *Provisioning and connecting to an Azure SQL database using PowerShell* recipe of *Chapter 5, Configuring and Securing Azure SQL Database*.

How to do it...

The steps for this recipe are as follows:

1. Execute the following PowerShell command to create an Azure Automation account:

```
#Create an Azure automation account
$automation = New-AzAutomationAccount -ResourceGroupName packtadesql -Name azadeautomation -Location eastus -Plan Basic
```

2. Execute the following command to create an Automation runbook of the PowerShell workflow type:

```
#Create a new automation runbook of type PowerShell
workflow
$runbook = New-AzAutomationRunbook -Name rnscalesql
-Description "Scale up sql azure when CPU is 40%" -Type
PowerShellWorkflow -ResourceGroupName packtadesql
-AutomationAccountName $automation.AutomationAccountName
```

3. Execute the following command to create Automation credentials. The credentials are passed as a parameter to the runbook and are used to connect to the Azure SQL database from the runbook:

```
#Create automation credentials.  
$sqladminpassword = ConvertTo-SecureString 'Sql@  
Server@1234' -AsPlainText -Force  
$sqladmincredential = New-Object System.Management.  
Automation.PSCredential('sqladmin', $sqladminpassword)  
$creds = New-AzAutomationCredential -Name sqlcred  
-Description "sql azure creds" -ResourceGroupName  
packtadesql -AutomationAccountName $automation.  
AutomationAccountName -Value $sqladmincredential
```

4. Go to portal.azure.com. Under **All resources**, search for and open the azadeautomation automation:

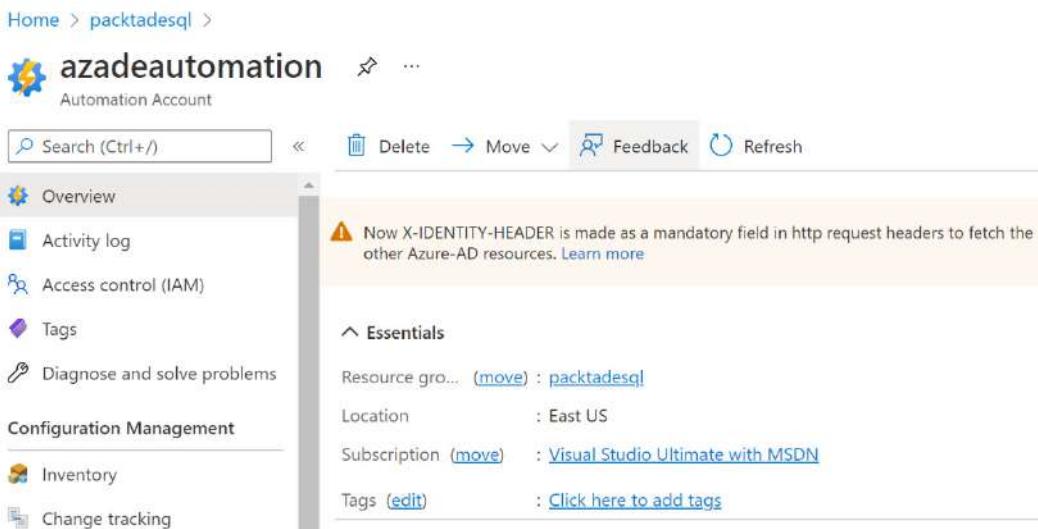


Figure 6.21 – Open Automation account

5. Search for and click on **Modules**. Click on the **Browse gallery** button:

The screenshot shows the 'azadeautomation' Automation Account's Modules page. At the top, there is a search bar with 'Modules' typed into it, and a 'Browse gallery' button highlighted with a red box. Below the search bar, there is a 'Shared Resources' section with a 'Modules' button also highlighted with a red box. The main area displays a table with one row: 'AuditPolicyDsc' (Status: Available, Type: Default). There are buttons for 'Add a module' and 'Update Az Modules' at the top of the table.

Figure 6.22 – Module installation in an Automation account

6. Search for `sqlServer`. Select the **SqlServer** module and hit the **Select** button on the next screen:

The screenshot shows the 'Browse Gallery' screen with a search bar containing 'sqlServer', which is also highlighted with a red box. The 'SqlServer' module is listed with a description: 'This module allows SQL Server developers, administrators and business intelligence professionals to automate database development and server administration, as well as both multidimensional and tabular cube processing.' It has several tags listed below it: SQL, SqlServer, SQLPS, Databases, SqlAgent, Jobs, SSAS, AnalysisServices, TabularCubes, SSIS, ExtendedEvents, xEvents, VulnerabilityAssessment, DataClassification, and PSMODULE.

Figure 6.23 – An Azure Automation account SqlServer module installation

7. Set **Runtime version** to **7.1** on the **Add a module** screen and hit the **Import** button:

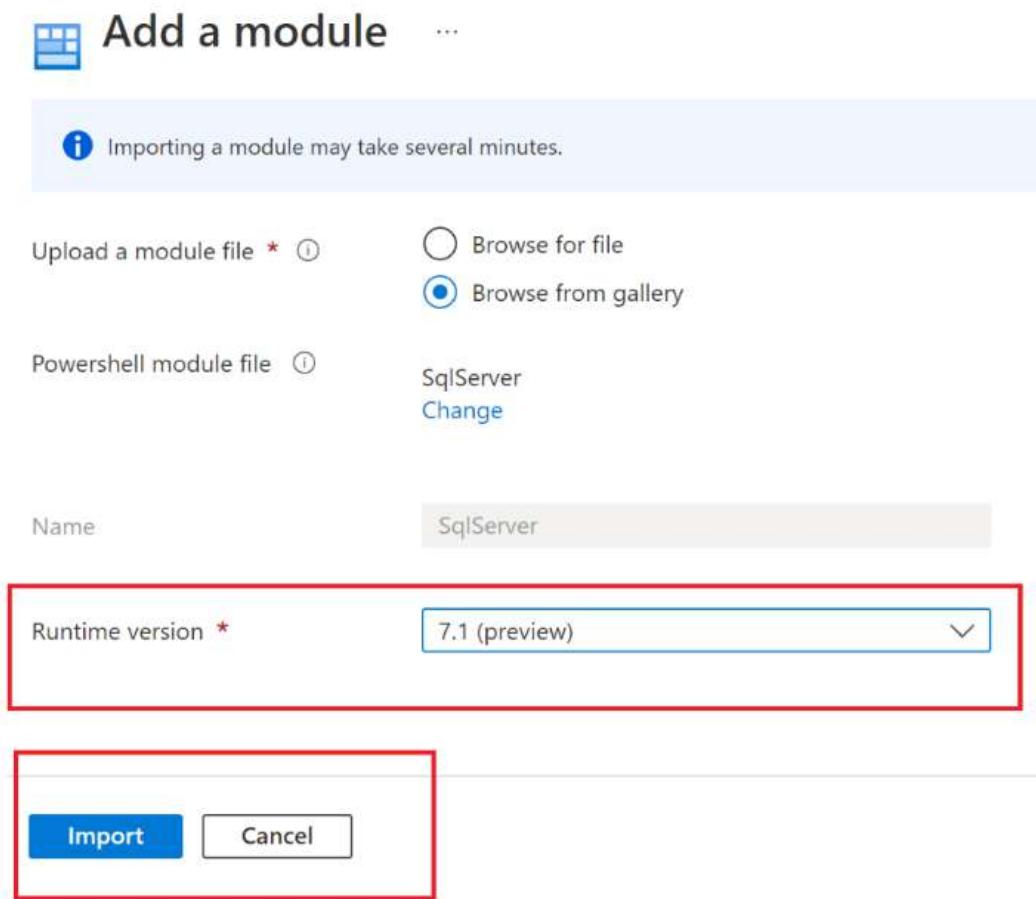


Figure 6.24 – Importing the SqlServer module

8. The next step is to edit the runbook and provide the PowerShell script to modify the service tier of an Azure SQL database. To do that, open <https://portal.azure.com> and log in to your Azure account. Under **All resources**, search for and open the **azadeautomation** Automation account:

The screenshot shows the Azure portal interface for an Automation Account named 'azadeautomation'. The left sidebar includes links for Overview, Activity log, Access control (IAM), Tags, Diagnose and solve problems, Configuration Management, Inventory, and Change tracking. The main content area displays the 'Essentials' section with the following details:

Resource group	(move) : packtadesql	Subscription ...	
Location	: East US	Status	: Active
Subscription	(move) : Visual Studio Ultimate with MSDN	Last modified	: 1/22/2022, 10:43 AM
Tags	(edit) : Click here to add tags		

Figure 6.25 – Opening the Azure Automation account

9. On the Azure Automation page, locate and select **Runbooks**:

The screenshot shows the 'Runbooks' page within the 'azadeautomation' Automation Account. The left sidebar has a link for Runbooks selected. The main area features a search bar with 'runbook', buttons for 'Create a runbook', 'Import a runbook', 'Browse gallery', and 'Learn more', and filters for 'Runbook type: All' and 'Authoring Status: All'. A table lists the runbooks:

Name	Authoring status	Runbook type
rnscalesql	New	PowerShell Workflow

Figure 6.26 – Opening Runbooks in Azure Automation

10. Select the **rnscalesql** runbook to open the **Runbook** page. On the **Runbook** page, select **Edit**:

The screenshot shows the Azure portal interface for managing a runbook. At the top, there's a breadcrumb navigation: Home > packtadesql > azadeautomation >. Below it, the title is 'rnsql (azadeautomation/rnsql)' with a 'Runbook' icon. The main area has a search bar and navigation buttons for Start, View, and Edit (which is highlighted with a red box). On the left, there's a sidebar with 'Overview' selected, followed by Activity log, Tags, and Diagnose and solve problems. The 'Essentials' section displays resource group (packtadesql), account (azadeautomation), location (East US), and subscription (Visual Studio Ultimate with MSDN). Below that is a 'Resources' section with a 'Jobs' link and a 'Tags' section with a 'Click here to add tags' link.

Figure 6.27 – Editing the runbook in your Azure Automation account

11. On the **Edit PowerShell Workflow Runbook** page, copy and paste the following PowerShell code into the canvas. The following section of the code receives parameters passed by the webhook:

```
workflow rnsql
{
    param
    (
        # Name of the Azure SQL Database server (Ex:bzb98er9bp)
        [parameter(Mandatory=$true)]
        [string] $SqlServerName,
        # Target Azure SQL Database name
        [parameter(Mandatory=$true)]
        [string] $DatabaseName,
        # When using in the Azure Automation UI, please enter
        # the name of the credential asset for the "Credential"
        # parameter
        [parameter(Mandatory=$true)]
        [PSCredential] $Credential
    )
}
```

The following section of the code uses the parameters passed and establishes a connection with the database:

```
inlinescript
{
$ServerName = $Using:SqlServerName + ".database.windows.
.net"
$db = $Using:DatabaseName
$UserId = $Using:Credential.UserName
$Password = ($Using:Credential).GetNetworkCredential()..
Password
$MasterDatabaseConnection = New-Object System.Data.
SqlClient.SqlConnection
$MasterDatabaseConnection.ConnectionString = "Server
= $ServerName; Database = Master; User ID = $UserId;
Password = $Password;"
$MasterDatabaseConnection.Open();
```

The following section of the code uses the ALTER DATABASE command and modifies the service tier of the Azure SQL Database to Standard S0:

```
$MasterDatabaseCommand = New-Object System.Data.
SqlClient.SqlCommand
$MasterDatabaseCommand.Connection
=$MasterDatabaseConnection
$MasterDatabaseCommand.CommandText = "ALTER DATABASE $db
MODIFY (EDITION = 'Standard', SERVICE_OBJECTIVE = 'S0');"
$MasterDbResult = $MasterDatabaseCommand.
ExecuteNonQuery();
}
```

12. Click **Save**, and then click **Publish** to publish the runbook:

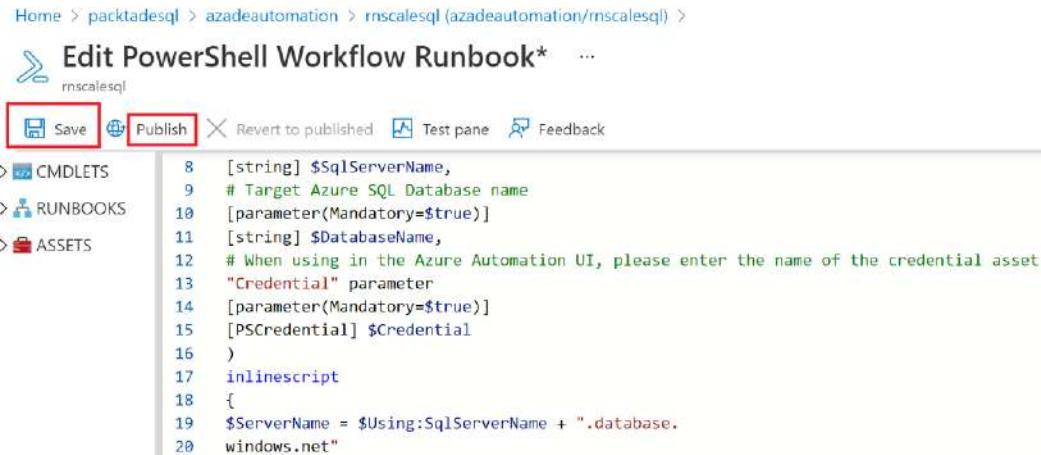


Figure 6.28 – Saving and publishing the runbook

13. The next step is to create a webhook to trigger the runbook. Execute the following PowerShell command to create the webhook:

```

# define the runbook parameters
$Params = @
{
    "SQLSERVERNAME"="azadesqlserver"; "DATABASENAME"
    ="azadesqlldb"; "CREDENTIAL"="sqlcred"
}
# Create a webhook
$expiry = (Get-Date).AddDays(1)
New-AzAutomationWebhook -Name rnscaleazure -RunbookName
$runbook.Name -Parameters $Params -ResourceGroupName
packtadesql -AutomationAccountName $automation.
AutomationAccountName -IsEnabled $true -ExpiryTime
$expiry

```

Note

When defining \$Params, you may want to change the default values mentioned here if you have a different Azure SQL Server, database, and credential values.

You should get an output as shown in the following screenshot:

```

Windows PowerShell
PS C:\Users\navenkat> # Define the runbook parameters
PS C:\Users\navenkat> $Params = @{"SQLSERVERNAME"="azadesqldb"; "DATABASENAME"="azadesql"; "CREDENTIAL"="sqlcred"}
PS C:\Users\navenkat> # Create a webhook
PS C:\Users\navenkat> $expiry = (Get-Date).AddDays(1)
PS C:\Users\navenkat> New-AzAutomationWebhook -Name rnscaleazure -RunbookName $runbook.Name -Parameters $Params -ResourceGroupName packtade
sql -AutomationAccountName $automation.AutomationAccountName -IsEnabled $true -ExpiryTime $expiry

Confirm
For security purposes, the URL of the created webhook will only be viewable in the output of this command. No other commands will return the webhook URL. Make sure to copy down the webhook URL from this command's output before closing your PowerShell session, and to store it securely.

[Y] Yes [N] No [S] Suspend [?] Help (default is "Y"): y

ResourceGroupName      : packtadesql
AutomationAccountName : azadeautomation
Name                  : rnscaleazure
CreationTime           : 22/1/2022 12:25:26 pm +08:00
Description             :
ExpiryTime             : 23/1/2022 12:25:20 pm +08:00
IsEnabled               : True
LastInvokedTime        : 1/1/0001 12:00:00 am +00:00
LastModifiedTime       : 22/1/2022 12:25:26 pm +08:00
Parameters              : {Credential, sqlServerName, databaseName}
RunbookName             : rnscalesql
WebhookURI             : https://e8c8271a-63e3-4bb7-b8d4-546f01d142f5.webhook.eus.azure-automation.net/webhooks?token=HE7yR07xdgbSW6Zz08LnE
GaOwv5h%2bVuDRIGEtIQdq9A%3d
HybridWorker            :

```

Figure 6.29 – Creating a webhook

Copy and save the WebhookURI value for later use.

14. The next step is to create an alert for an Azure SQL database that, when triggered, will call the webhook URI. Execute the following query to create an alert action group receiver:

```

#Create action group receiver
$whr = New-AzActionGroupReceiver -Name agrscalesql
-WebhookReceiver -ServiceUri "https://e8c8271a-63e3-
4bb7-b8d4-546f01d142f5.webhook.eus.azure-automation.net/
webhooks?token=HE7yR07xdgbSW6Zz08LnE
GaOwv5h%2bVuDRIGEtIQdq9A%3d"

```

Note

Replace the value of the ServiceUri parameter with your webhook URI from the previous step.

15. Execute the following query to create an action group with an action receiver as defined by the preceding command:

```

#Create a new action group
$ag = Set-AzActionGroup -ResourceGroupName packtade -Name
ScaleSQLAzure -ShortName scaleazure -Receiver $whr

```

16. Execute the following query to create an alert condition to trigger the alert:

```
#define the alert trigger condition
$condition = New-AzMetricAlertRuleV2Criteria -MetricName
"cpu_percent" -TimeAggregation maximum -Operator
greaterthan -Threshold 40 -MetricNamespace "Microsoft.
Sql/servers/databases"
```

The condition defines that the alert should trigger when the metric CPU percentage is greater than 40%.

17. Execute the following query to create an alert on the Azure SQL database:

```
#Create the alert with the condition and action defined
in previous steps.

$rid = (Get-AzSqlDatabase -ServerName azadesqlserver
-ResourceGroupName packtadesql -DatabaseName azadesqldb).ResourceId
Add-AzMetricAlertRuleV2 -Name monitorcpu
-ResourceGroupName packtadesql -WindowSize 00:01:00
-Frequency 00:01:00 -TargetResourceId $rid -Condition
$condition -Severity 1 -ActionGroupId $ag.id
```

You should get an output as shown in the following screenshot:

```
PS C:\Users\havenvkat> $rid = (Get-AzSqlDatabase -ServerName azadesqlserver -ResourceGroupName packtadesql -DatabaseName azadesqldb).ResourceId
WARNING: Upcoming breaking changes in the cmdlet 'Get-AzSqlDatabase' :
- The output type 'Microsoft.Azure.Commands.Sql.Database.Model.AzureSqlDatabaseModel' is changing
- The following properties in the output type are being deprecated : 'BackupStorageRedundancy'
- The following properties are being added to the output type : 'CurrentBackupStorageRedundancy' 'RequestedBackupStorageRedundancy'
- The change is expected to take effect from the version : '3.0.0'
Note : Go to https://aka.ms/azps-changewarnings for steps to suppress this breaking change warning, and other information on breaking changes in Azure PowerShell.
PS C:\Users\havenvkat> Add-AzMetricAlertRuleV2 -Name monitorcpu -ResourceGroupName packtadesql -WindowSize 00:01:00 -Frequency 00:01:00 -TargetResourceId $rid -Condition $condition -Severity 1 -ActionGroupId $ag.id
WARNING: 7:45:23 am - The namespace for all the model classes will change from Microsoft.Azure.Management.Monitor.Management.Models to Microsoft.Azure.Management.Monitor.Models in releases.
WARNING: 7:45:23 am - The namespace for output classes will be uniform for all classes in future releases to make it independent of modifications in the model classes.

Description      : This new Metric alert rule was created from Powershell version: 3.0.0
Severity        : 1
Enabled          : True
Scopes           : (/subscriptions/[REDACTED]/resourceGroups/packtadesql/providers/Microsoft.Sql/servers/azadesqlserver/databases/azadesqldb)
EvaluationFrequency: 00:01:00
WindowSize       : 00:01:00
TargetResourceType: 
TargetResourceRegion: 
Criteria         : Microsoft.Azure.Management.Monitor.Models.MetricAlertSingleResourceMultipleMetricCriteria
AutoMitigate     : True
Actions          : {}
LastUpdatedTime: 
IsMigrated       : 
Id               : /subscriptions/[REDACTED]/resourceGroups/packtadesql/providers/Microsoft.Insights/metricAlerts/monitorcpu
Name             : monitorcpu
Type             : Microsoft.Insights/metricAlerts
Location         : global
Tags             : 
Kind             : 
Etag             :
```

Figure 6.30 – Creating the alert

The preceding command creates an Azure SQL database alert. The alert is triggered when the `cpu_percent` metric is greater than 40% for more than 1 minute. When the alert is triggered, as defined in the action group, the webhook is called. The webhook in turn runs the runbook. The runbook modifies the service tier of the database to Standard S0.

Go to `portal.azure.com`, find `azadesqlserver`, and go to the `azadesqlldb` database. Click on the **Alerts** option on the left-hand side. Click **Alert rules**:

The screenshot shows the Azure portal interface for managing alert rules. The URL in the address bar is `Home > azadesqlldb (azadesqlserver/azadesqlldb) > azadesqlserver > azadesqlldb (azadesqlserver/azadesqlldb)`. The main title is **azadesqlldb (azadesqlserver/azadesqlldb) | Alerts**. On the left, there's a navigation menu with **Monitoring** and **Alerts** (which is highlighted with a red box). At the top, there are buttons for **Alert** (with a red box), **Create**, **Alert rules** (also with a red box), **Action groups**, **Alert processing rules**, **Columns**, **Refresh**, and **Export to CSV**. Below these are search and filter fields: **Search** (with a red box), **Resource name : azadesqlserver/azadesqlldb**, and **Time range : Past 24 hours**. The main area shows alert counts for different severities: **Total alerts: 0**, **Critical: 0**, **Error: 0**, **Warning: 0**, **Informational: 0**, and **Verbose: 0**. At the bottom, there are sorting options: **Name ↑↓**, **Severity ↑↓**, **Alert condition ↑↓**, and **User response ↑↓**.

Figure 6.31 – Alert rules

You will notice that an alert condition called **monitorcpu** has been created:

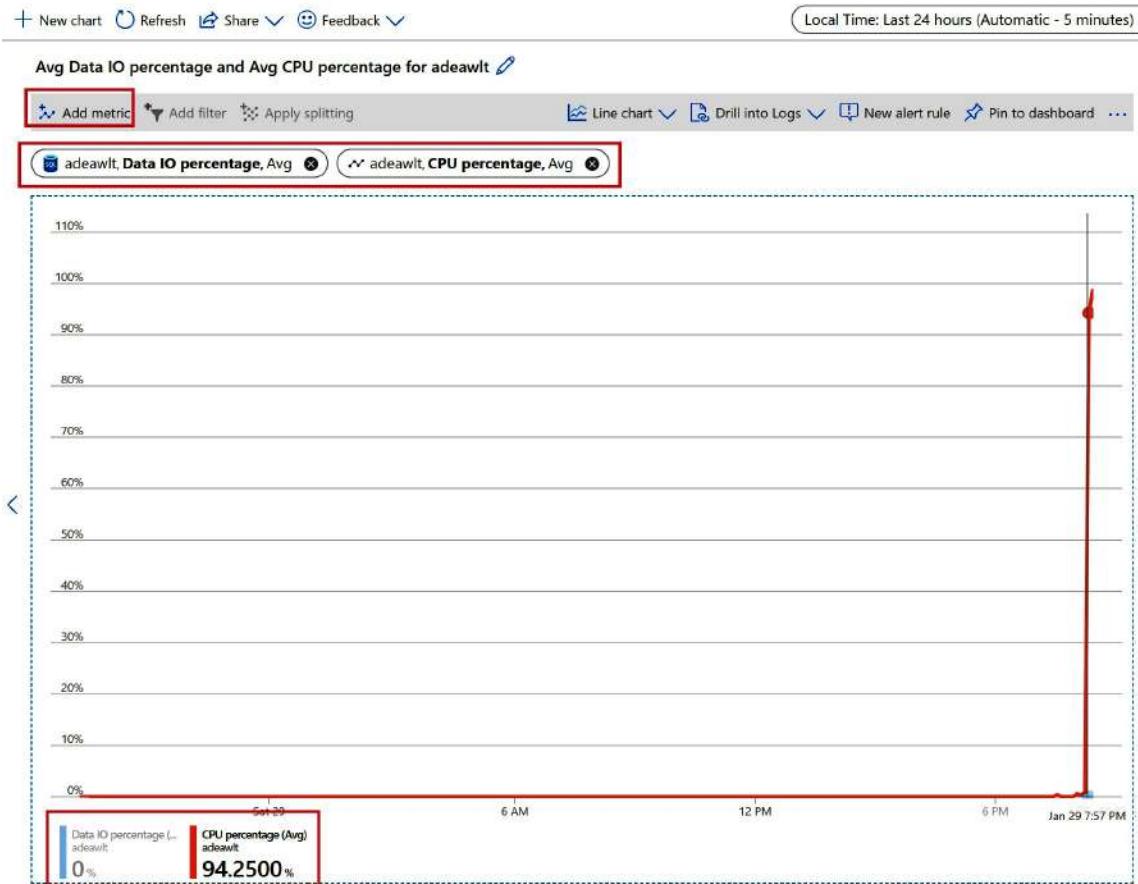


Figure 6.32 – CPU alert

18. To ensure that the runbook is able to connect to Azure SQL Database, execute the following PowerShell command to allow all connections from Azure services to connect to the Azure SQL database server named azadesqlserver:

```
New-AzSqlServerFirewallRule -FirewallRuleName
    "AllowAllAzureIPs" -StartIpAddress "0.0.0.0"
    EndIpAddress "0.0.0.0" -ServerName azadesqlserver
    -ResourceGroupName packtadesql
```

19. The PowerShell command will return a result as shown here:

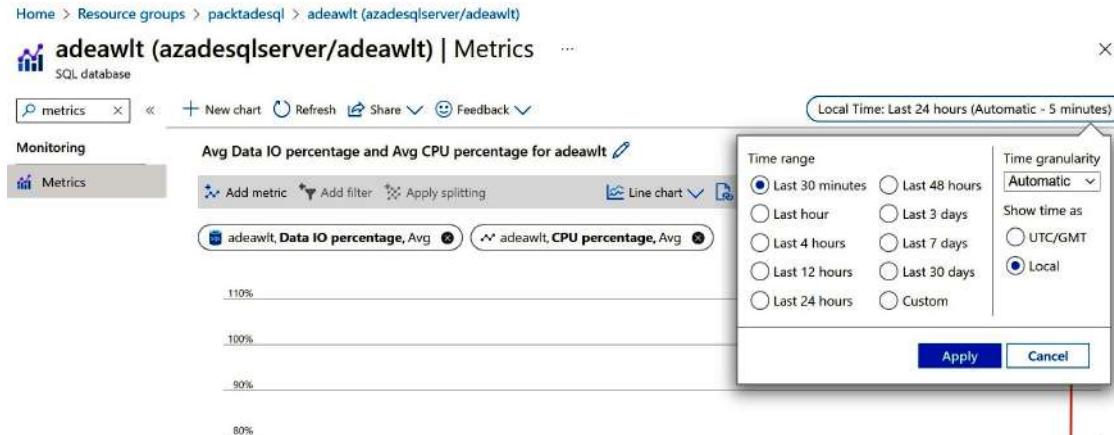


Figure 6.33 – Allow all Azure connections

20. To see the alert in action, connect to the Azure SQL database and execute the following query to simulate high CPU usage:

```
--query to simulate high CPU usage
While(1=1)
Begin
Select cast(a.object_id as nvarchar(max)) from sys.
objects a, sys.objects b,sys.objects c, sys.objects d
End
```

As soon as the alert condition is triggered, the webhook is called and the database service tier is modified to Standard S0. You can check that the alert was triggered by navigating to the **azadesqlldb** database on the Azure portal and clicking on the **Alerts** option. **Fired time** confirms that a CPU spike was detected and the alert was fired:



Figure 6.34 – Alert triggered

How it works...

To configure automatic scaling for an Azure SQL database, we create an Azure Automation runbook. The runbook specifies the PowerShell code to modify the service tier of an Azure SQL database.

We create a webhook to trigger the runbook. We create an Azure SQL database alert and define the alert condition as `cpu_percent` metric is greater than 40% for at least 1 minute. In the alert action, we call the webhook defined earlier.

When the alert condition is reached, the webhook is called, which in turn executes the runbook, resulting in the Azure SQL Database service tier change.

Monitoring an Azure SQL database using the Azure portal

Azure SQL Database has built-in monitoring features, such as Query Performance Insights, performance overview, and diagnostic logging. In this recipe, we'll learn how to use the monitoring capabilities using the Azure portal.

Getting ready

We'll use PowerShell to create an Azure SQL database, so open a PowerShell window and log in to your Azure account by executing the `Connect-AzAccount` command.

We'll use the Azure portal to monitor the Azure SQL database. Open <https://portal.azure.com> and log in to your Azure account.

How to do it...

First, let's execute a sample workload.

Creating an Azure SQL database and executing a sample workload

The steps are as follows:

1. Execute the following PowerShell command to create an Azure SQL database using the `AdventureWorksLT` sample database:

```
# create the resource group
New-AzResourceGroup -Name packtadesql -Location "central
us" -force
#create credential object for the Azure SQL Server admin
credential
$sqladminpassword = ConvertTo-SecureString 'Sql@
Server@1234' -AsPlainText -Force
```

```
$sqladmincredential = New-Object System.Management.Automation.PSCredential ('sqladmin', $sqladminpassword)
# create the Azure SQL Server
New-AzSqlServer -ServerName azadesqlserver
-SqlAdministratorCredentials $sqladmincredential
-Location "central us" -ResourceGroupName packtadesql
#Create the SQL Database
New-AzSqlDatabase -DatabaseName adeawlt -Edition basic
-ServerName azadesqlserver -ResourceGroupName packtadesql
-SampleName AdventureWorksLT
```

2. Execute the following command to add the client IP to the Azure SQL Server firewall:

```
$clientip = (Invoke-RestMethod -Uri https://ipinfo.io/json).ip
New-AzSqlServerFirewallRule -FirewallRuleName "home"
-StartIpAddress $clientip -EndIpAddress $clientip
-ServerName azadesqlserver -ResourceGroupName packtadesql
```

3. Download the `workload.sql` file from <https://github.com/PacktPublishing/Azure-Data-Engineering-Cookbook-2nd-edition/blob/main/Chapter06/workload.sql> and execute the following command to run a workload against the Azure SQL Database. Ensure to edit the script to set your correct local path for input and output parameters:

```
sqlcmd -S azadesqlserver.database.windows.net -d adeawlt
-U sqladmin -P Sql@Server@1234 -i "C:\ADECookbook\Chapter06\workload.sql" > "C:\ADECookbook\Chapter06\workload_output.txt"
```

It can take 4 to 5 minutes for the workload to complete. You can execute the preceding command multiple times; however, you should run it at least once.

Monitoring Azure SQL Database metrics

The steps are as follows:

1. In the Azure portal, navigate to **All resources | azadesqlserver** and find the **adeawlt** database. Search for and open **Metrics**:

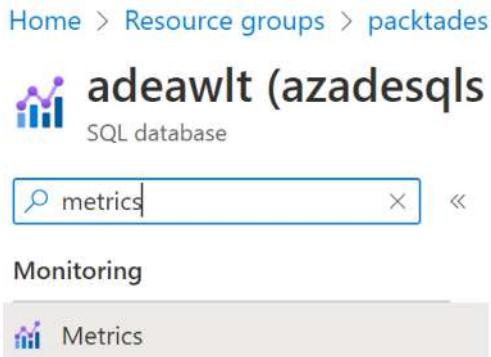


Figure 6.35 – Opening the Metrics section in the Azure portal

The **Metrics** page allows you to monitor different available metrics over time.

2. To select metrics, click **Add metric** | **CPU percentage** | **Data IO percentage**:

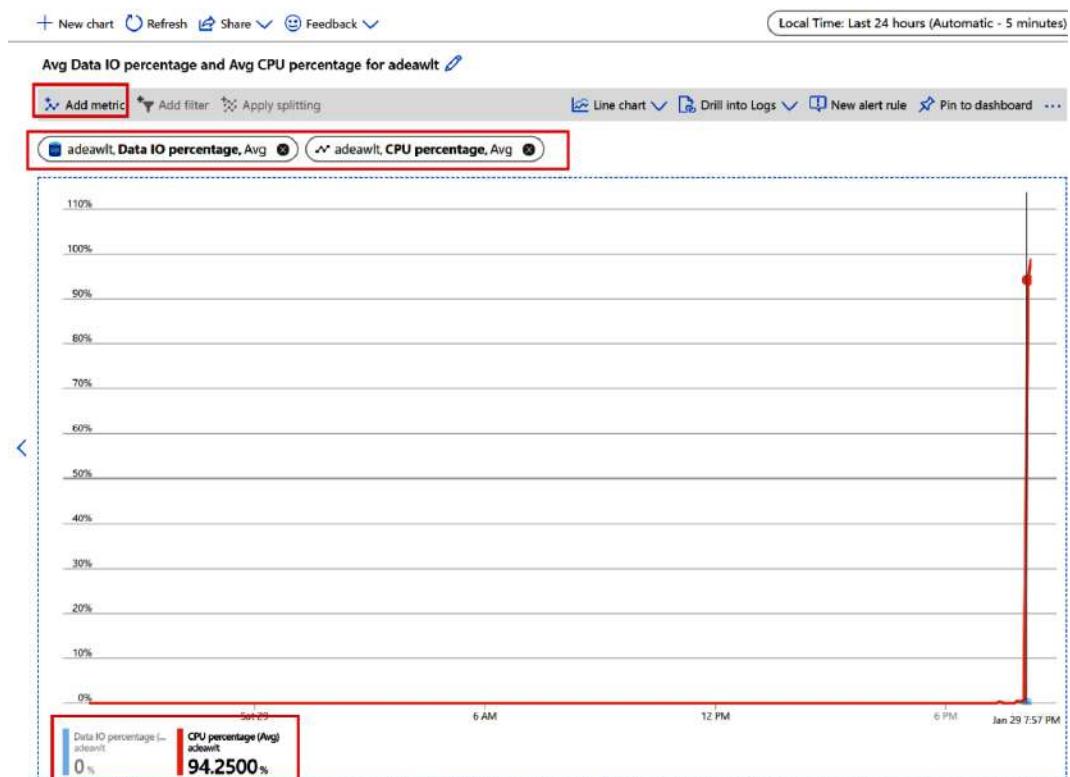


Figure 6.36 – Monitoring the metrics for a SQL database

We can select the metrics we are interested in monitoring and use the **Pin to dashboard** feature to pin the chart to the portal dashboard. We can also create an alert rule from the **Metrics** page by clicking on **New alert rule**. We can select a time range to drill down to specific times or investigate spikes in the chart.

3. To select a time range, select the **Time range** dropdown in the top-right corner of the **Metrics** page and select the desired time range:

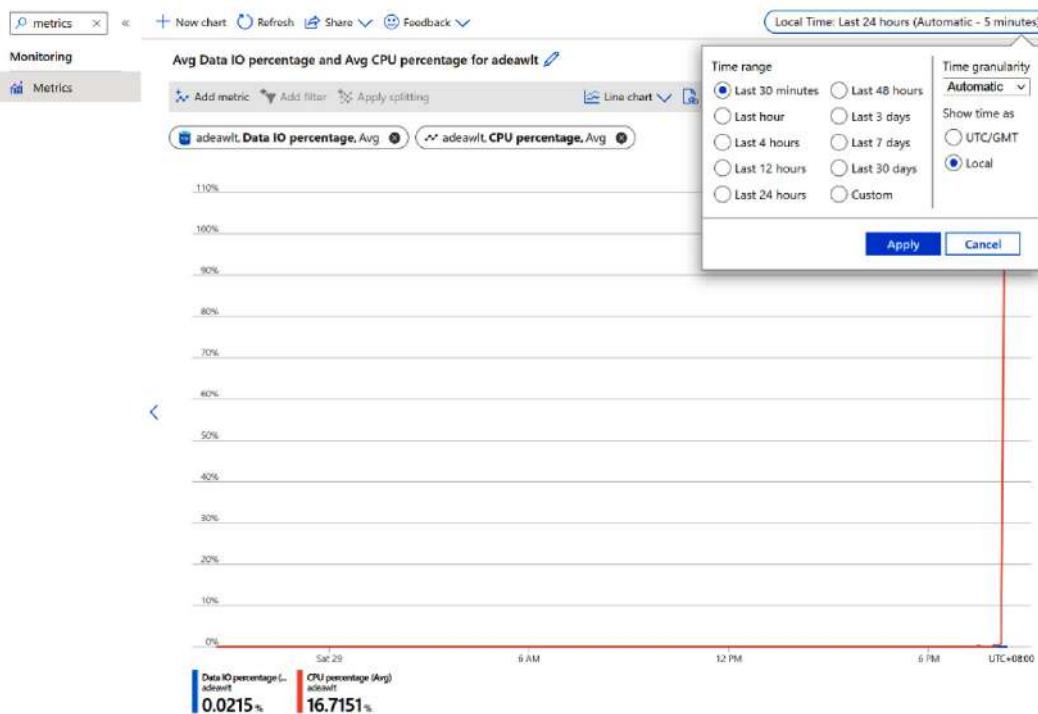


Figure 6.37 – Selecting a time range to monitor

Using Query Performance Insight to find resource-consuming queries

Query Performance Insight is an intelligent performance feature that allows us to find any resource-consuming and long-running queries. The steps are as follows:

1. In the Azure portal, navigate to **All resources | azadesqlserver** and find the **adeawlt** database. Find and open **Query Performance Insight**:

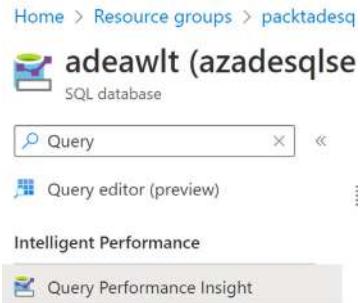


Figure 6.38 – Selecting Query Performance Insight for the SQL database

2. On the **Query Performance Insight** page, observe that there are three tabs: **Resource consuming queries**, **Long running queries**, and **Custom**. We can select resource-consuming queries by **CPU**, **Data IO**, and **Log IO**. The table at the bottom of the page lists the query IDs by CPU consumption. Click on the color-coded box in the **QUERY ID** column to get the query text:

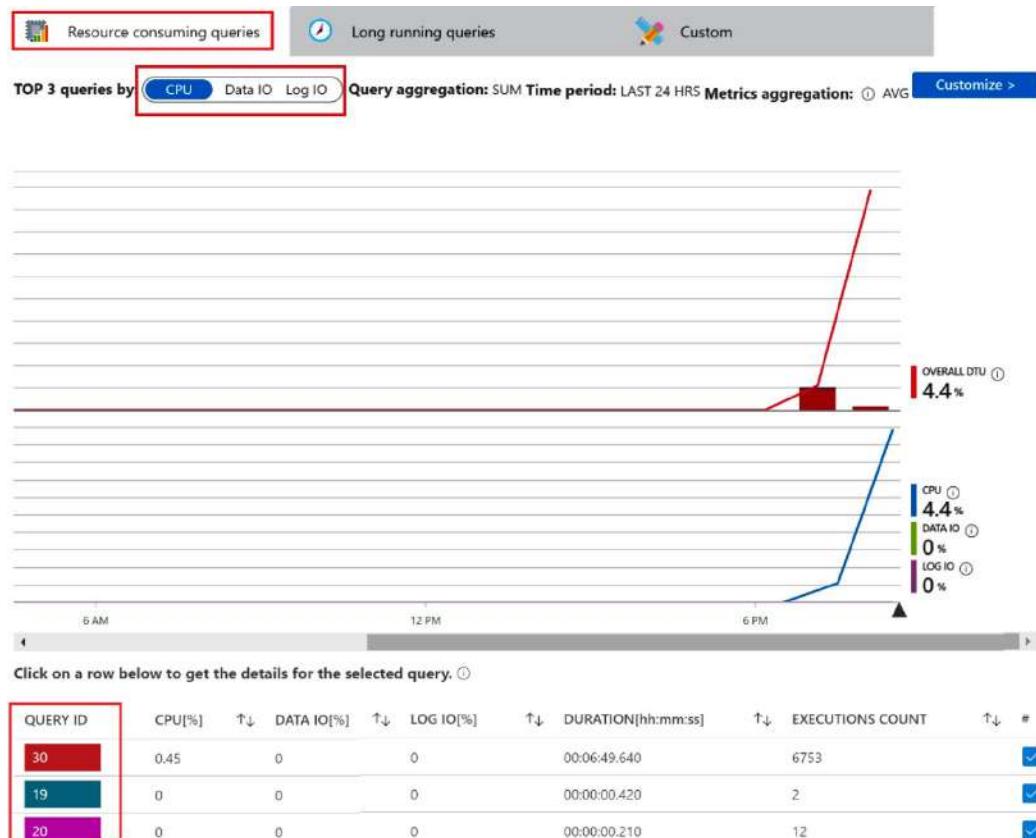


Figure 6.39 – Monitoring queries for the SQL database

3. The query details for query ID **30** are shown in the following screenshot:

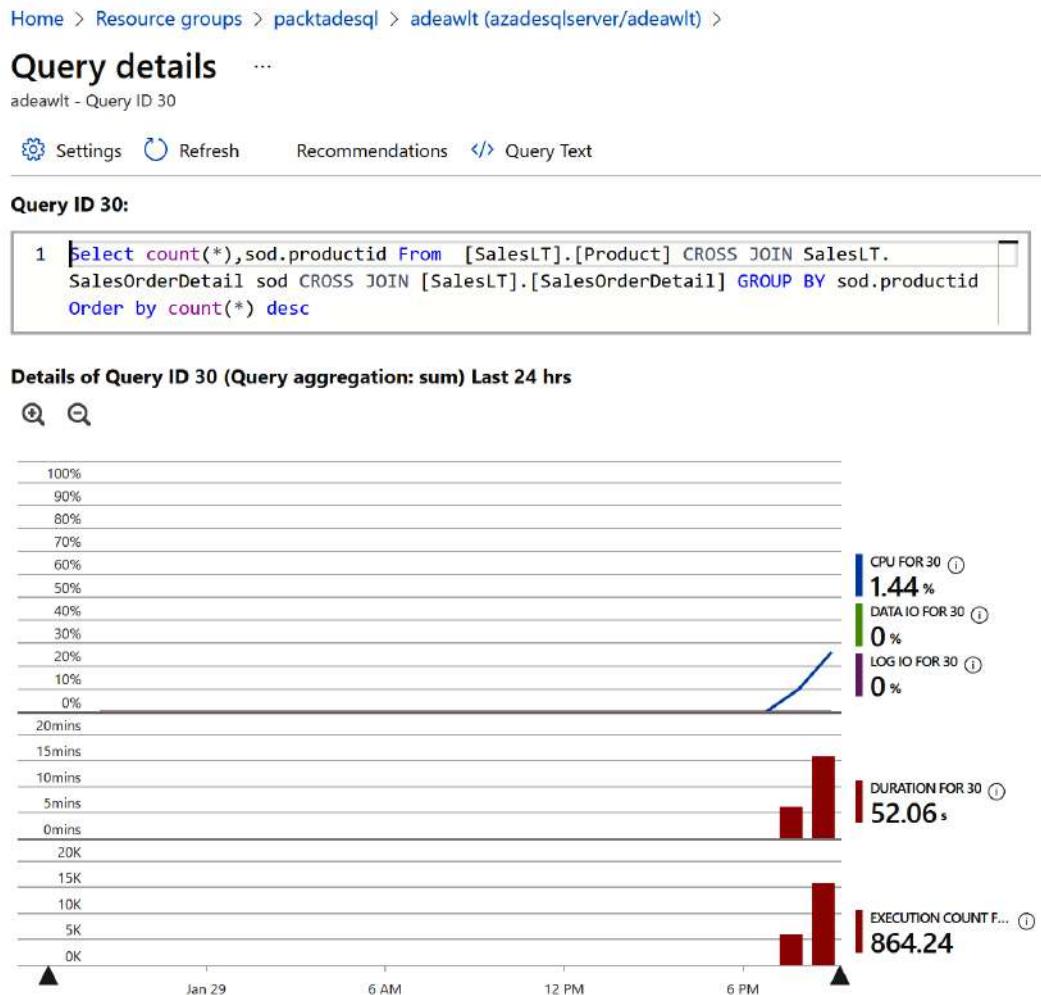


Figure 6.40 – Viewing the query details

We can look at the query text and optimize it for better performance.

4. The **Custom** tab allows us to select resource-consuming queries by duration and execution count. We can also specify a custom time range, the number of queries, and the query and metric aggregation:

The screenshot shows the 'adeawlt (azadesqlserver/adeawlt) | Query Performance Insight' page. At the top, there's a navigation bar with 'Home > Resource groups > packtadessql > adeawlt (azadesqlserver/adeawlt)'. Below it is a toolbar with 'Query', 'Reset settings', 'Refresh', 'Recommendations', 'Getting started', and 'Feedback'. A red box highlights the 'Custom' button in the top right corner of the toolbar.

The main area is titled 'Resource consuming queries' and has tabs for 'Resource consuming queries' (selected), 'Long running queries', and 'Custom'. Under 'Intelligent Performance', there's a section for 'Query Performance Insight' with dropdown menus for 'Metric type' (set to 'Duration'), 'Time period' (set to 'Last 24 hrs'), 'Number of queries' (set to '5'), 'Query aggregation' (set to 'sum'), and 'Metrics aggregation' (set to 'avg'). A red box highlights the 'Duration' option under 'Metric type'. Below this, there are buttons for 'CPU', 'Data IO', 'Log IO', and 'Execution count'. A tooltip at the bottom of the dropdown says 'PU Query aggregation: SUM Time period: LAST 24 HRS Metrics aggregation: AVG'.

Figure 6.41 – Providing custom monitoring configuration

5. Select the options and click the **Go** button to refresh the chart as per the custom settings. We will get the top three queries by duration:

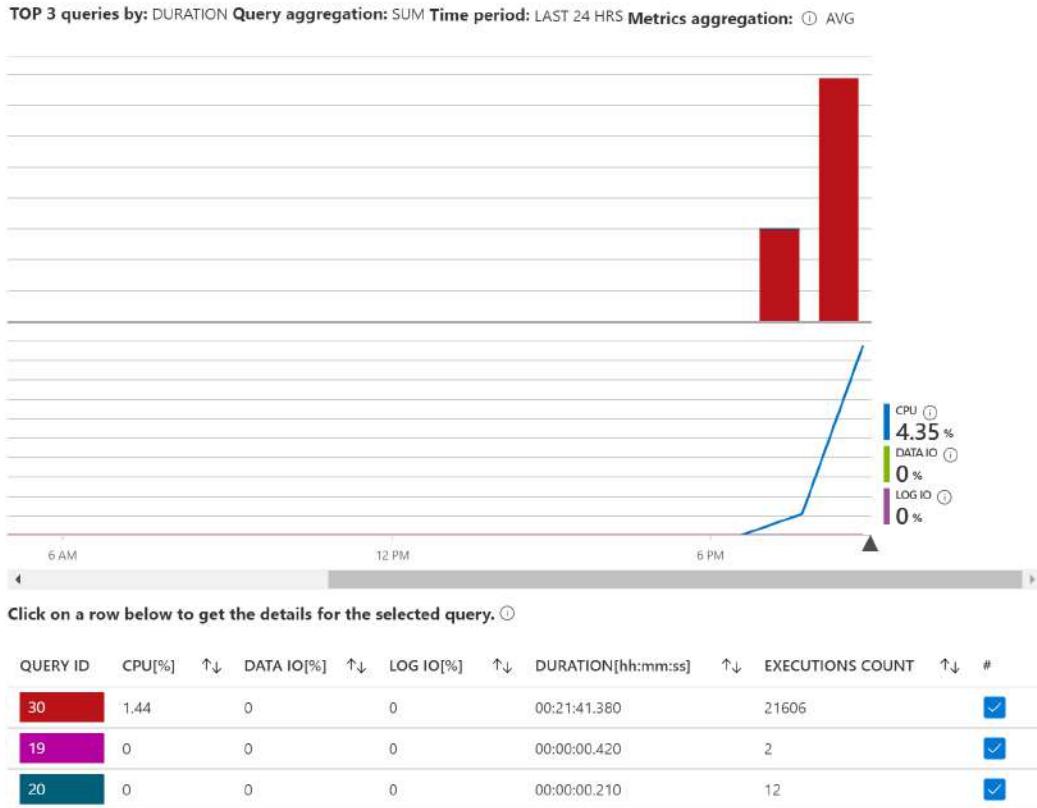


Figure 6.42 – Viewing the long-running queries list

We can further look into the query text and other details by selecting the query ID.

Monitoring an Azure SQL database using diagnostic settings

In addition to metrics and Query Performance Insight, we can also monitor an Azure SQL database by collecting diagnostic logs. The diagnostic logs can be sent to a Log Analytics workspace or Azure Storage, or can be streamed to Azure Event Hubs.

Log Analytics is an Azure service that can store diagnostic data from various Azure services and allows us to analyze the data in one place. Let's create a Log Analytics workspace to store the diagnostics details of the SQL database. The steps to create a Log Analytics workspace and configure the diagnostics for a SQL database are as follows:

1. Execute the following command to create a Log Analytics workspace named packtadesql1gw:

```
$ResourceGroup = "packtadesql"
$WorkspaceName = "packtadesql1gw"
$Location = "central us"
# Create the workspace
New-AzOperationalInsightsWorkspace -Location $Location
-Name $WorkspaceName -ResourceGroupName $ResourceGroup
```

2. To enable diagnostic logging using the Azure portal, navigate to **All resources | azadesqlserver | adeawlt**. Find and open **Diagnostic settings**:

The screenshot shows the Azure portal interface for managing diagnostic settings. At the top, the URL is [Home > Resource groups > packtadesql > adeawlt \(azadesqlserver/adeawlt\)](#). The main title is **adeawlt (azadesqlserver/adeawlt) | Diagnostic settings**. On the left, there's a sidebar with sections like **Diagnose and solve problems**, **Monitoring** (which is highlighted with a red box), **Support + troubleshooting**, and **Support + Troubleshooting**. The **Monitoring** section has a sub-section **Diagnostic settings** (also highlighted with a red box). Below this, there's a table with columns **Name**, **Storage account**, and **Event hub**. A message says **No diagnostic settings defined**. At the bottom of this section, there's a button **+ Add diagnostic setting** (highlighted with a red box). Below the table, a note says **Click 'Add Diagnostic setting' above to configure the collection of the following data:** followed by a bulleted list of data types: **SQLInsights**, **AutomaticTuning**, **QueryStoreRuntimeStatistics**, **QueryStoreWaitStatistics**, **Errors**, **DatabaseWaitStatistics**, **Timeouts**, **Blocks**, **Deadlocks**, **Basic**, **InstanceAndAppAdvanced**, and **WorkloadManagement**.

Figure 6.43 – Diagnostic settings

3. Click **+ Add diagnostic setting** to add a new diagnostic setting.
4. Select the categories to be included in the logs and their destination. Provide any name for the diagnostic setting. Select **Send to Log Analytics workspace** under **Destination details**. Select **packtadesqlgw** as the Log Analytics workspace to store the diagnostic log. Click **Save** to create the new diagnostic setting.

Home > Resource groups > packtadesql > adeawlt (azadesqlserver/adeawlt) >

Diagnostic setting ...

A diagnostic setting specifies a list of categories of platform logs and/or metrics that you want to collect from a resource, and one or more destinations that you would stream them to. Normal usage charges for the destination will occur. Learn more about the different log categories and contents of those logs

Diagnostic setting name *

sqldiagnostics

Logs

Category groups ⓘ

allLogs audit

Categories

SQLInsights
 AutomaticTuning
 QueryStoreRuntimeStatistics
 QueryStoreWaitStatistics
 Errors
 DatabaseWaitStatistics
 Timeouts
 Blocks
 Deadlocks

Destination details

Send to Log Analytics workspace

Subscription

Visual Studio Ultimate with MSDN

Log Analytics workspace

packtadesqlgw (central us)

Archive to a storage account
 Stream to an event hub
 Send to partner solution

Figure 6.44 – Selecting categories

Important Note

Diagnostic settings add an additional cost to the Azure SQL database. It may also take some time for the logs to be available after creating a diagnostic setting.

Automatic tuning in an Azure SQL database

Automatic tuning provides three features: **FORCE PLAN**, **CREATE INDEX**, and **DROP INDEX**. Automatic tuning can be enabled for an Azure SQL Server, in which case it's applied to all of the databases in that Azure SQL Server. Automatic tuning can be enabled for individual Azure SQL databases as well. The steps are as follows:

1. To enable automatic tuning, in the Azure portal, navigate to **All resources | azadesqlserver | adeawlt**. Find and select **Automatic tuning** under **Intelligent Performance**:

The screenshot shows the Azure portal interface for managing an Azure SQL database named 'adeawlt'. On the left, there's a sidebar with options like Home, Resource groups, and a search bar. The main area is titled 'adeawlt (azadesqlserver/adeawlt) | Automatic tuning'. A red box highlights the 'Apply' button at the top right. Below it, a message states: 'Azure SQL Database built-in intelligence automatically tunes your databases to optimize performance. Click here to learn more about automatic tuning.' Another red box highlights the 'Automatic tuning' section in the 'Intelligent Performance' menu on the left. The 'Configure the automatic tuning options' table has three rows: 'FORCE PLAN' (Desired state: ON, Current state: ON), 'CREATE INDEX' (Desired state: ON, Current state: OFF), and 'DROP INDEX' (Desired state: INHERIT, Current state: OFF). The 'CREATE INDEX' row is also highlighted with a red box.

Option	Desired state	Current state
FORCE PLAN	ON OFF INHERIT	ON Inherited from server
CREATE INDEX	ON OFF INHERIT	OFF Inherited from server
DROP INDEX	ON OFF INHERIT	OFF Inherited from server

Figure 6.45 – Automatic tuning in the SQL database

2. Enable the **CREATE INDEX** tuning option by clicking **ON** under the **Desired state** option.
3. Click **Apply** to save the configuration.

Important Note

It may take some time for the recommendation to show up.

The recommendations will show up in the performance recommendations in the **Intelligent Performance** section.

Configuring auditing for Azure SQL Database

SQL Auditing is a process in which key activities or changes in the database are recorded not only for monitoring but also for compliance to global security standards. In the following recipe, we will configure an audit for Azure SQL Database to record the database activities, store the details in Azure Log Analytics, and verify the recorded data.

Getting ready

Create an Azure SQL database and a Log Analytics workspace, as mentioned in the *Monitoring an Azure SQL database using the Azure portal* recipe of this chapter.

How to do it...

- In the Azure portal, navigate to **All resources** | **azadesqlserver** | **adeawlt**. Find **Auditing**:

The screenshot shows the Azure Log Analytics workspace interface. The left sidebar has sections for 'Logs', 'Activity log', 'Custom logs', 'General', and 'Workspace summary'. The 'Logs' section is highlighted with a red box. The main area shows a 'New Query' tab with the identifier 'packtadesqllgw'. Below it is a search bar and a 'Run' button, which is also highlighted with a red box. The query editor contains the following T-SQL code:

```

1 AzureDiagnostics
2 | where Category == 'SQLSecurityAuditEvents' and action_name_s == "BATCH_COMPLETED"
3 | project event_time_t,ResourceGroup,server_instance_name_s,database_name_s,statement_s,action_name_s,
server_principal_name_s,application_name_s,duration_milliseconds_d,response_rows_d,affected_rows_d
4 | sort by event_time_t desc

```

Figure 6.46 – Azure SQL Database Auditing

- Turn on **Azure SQL Auditing**. Enable **Log Analytics**. Pick the **packtadesqllgw** Log Analytics workspace, which was created in the *Monitoring an Azure SQL Database using the Azure portal* recipe of this chapter. Hit the **Save** button:

[Home](#) > [Resource groups](#) > [packtadesql](#) > [adeawlt \(azadesqlserver/adeawlt\)](#)

adeawlt (azadesqlserver/adeawlt) | Auditing

SQL database

Auditing Save Discard View audit logs Feedback

Security

Auditing

Info If Blob Auditing is enabled on the server, it will always apply to the database, regardless of the database settings.

[View server settings](#)

Info Server-level Auditing: **Disabled**

Azure SQL Auditing

Azure SQL Auditing tracks database events and writes them to an audit log in your Azure Storage account, Log Analytics workspace or Event Hub. [Learn more about Azure SQL Auditing](#)

Enable Azure SQL Auditing

Audit log destination (choose at least one):

Storage

Log Analytics

Subscription *

Visual Studio Ultimate with MSDN

Log Analytics *

packtadesqlgw(centralus)

Event Hub

Figure 6.47 – Configure Auditing

3. Connect to the database using SSMS and execute the following queries:

```
select ProductID from SalesLT.SalesOrderDetail where
OrderQty > 2
GO
Delete from SalesLT.SalesOrderDetail where ProductID
between 900 and 1000
GO
```

4. Go to portal.azure.com and then to **All Resources**. Find **packtadesqllgw**. Click on it to open the Log Analytics workspace:

The screenshot shows the Azure portal's 'All resources' blade. At the top, there are navigation links and a search bar containing 'packtadesqllgw'. Below the search bar are several filter buttons: 'Subscription == all', 'Resource group == all', 'Type == all', and a 'Logs' button. The main area displays a table with one record:

Name	Type
packtadesqllgw	Log Analytics workspace

Figure 6.48 – Open the Log Analytics workspace

5. Click **Logs** in the **General** section. Paste the following **Kusto Query Language (KQL)** script in the query window and hit the **RUN** button:

```
AzureDiagnostics
| where Category == 'SQLSecurityAuditEvents' and action_name_s == "BATCH COMPLETED"
| project event_time_t, ResourceGroup, server_instance_name_s, database_name_s, statement_s, action_name_s, server_principal_name_s, application_name_s, duration_milliseconds_d, response_rows_d, affected_rows_d
| sort by event_time_t desc
```

The following screenshot shows the script execution:

This screenshot shows the Azure Log Analytics workspace interface. On the left, there's a sidebar with sections like 'Settings', 'Custom logs', 'General', 'Workspace summary', and 'Logs'. The 'Logs' section is currently selected and highlighted with a red border. In the main area, there's a search bar labeled 'Logs' and a 'New Query 1+' button. Below that is a 'Run' button which is also highlighted with a red border. To the right of the run button are options for 'Feedback', 'Queries', 'Query explorer', 'Save', 'Share', 'New alert rule', and 'Export'. A time range filter 'Last 24 hours' is applied. The central part of the screen displays a KQL script:

```

1 AzureDiagnostics
2 | where Category == 'SQLSecurityAuditEvents' and action_name_s == "BATCH COMPLETED"
3 | project event_time_t,ResourceGroup,server_instance_name_s,database_name_s,statement_s,action_name_s,
server_principal_name_s,application_name_s,duration_milliseconds_d,response_rows_d,affected_rows_d
4 | sort by event_time_t desc

```

Figure 6.49 – Running the KQL script in the Log Analytics workspace

6. Scroll to the right on the **Results** screen to verify whether the fired queries are being recorded by the audit in the Log Analytics workspace. The fired DELETE and SELECT script should be listed in the **statement_s** column:

This screenshot shows the results of the KQL query execution. At the top, there are tabs for 'Results' (which is selected) and 'Chart'. Below that is a status message 'Completed. Showing results from the last 24 hours.' and a timestamp '00:01.2' with a note '9 records'. The results table has columns: 'statement_s', 'action_name_s', 'server_principal_name_s', and 'application_name_s'. The first two rows of the table are highlighted with a red border:

statement_s	action_name_s	server_principal_name_s	application_name_s
Delete from SalesLT.SalesOrderDetail where ProductID between...	BATCH COMPLETED	sqladmin	Microsoft SQL Ser...
select ProductID from SalesLT.SalesOrderDetail where OrderQty...	BATCH COMPLETED	sqladmin	Microsoft SQL Ser...
SELECT @@SPID;	BATCH COMPLETED	sqladmin	Microsoft SQL Ser...
select ProductID from SalesLT.SalesOrderDetail where OrderQty...	BATCH COMPLETED	sqladmin	Microsoft SQL Ser...
SELECT @@SPID;	BATCH COMPLETED	sqladmin	Microsoft SQL Ser...
DECLARE @edition sysname; SET @edition = cast(SERVERPROP...	BATCH COMPLETED	sqladmin	Microsoft SQL Ser...
select @@spid; select SERVERPROPERTY('ProductLevel'); SELEC...	BATCH COMPLETED	sqladmin	Microsoft SQL Ser...
SET ROWCOUNT 0 SET TEXTSIZE 2147483647 SET NOCOUNT O...	BATCH COMPLETED	sqladmin	Microsoft SQL Ser...
DECLARE @edition sysname; SET @edition = cast(SERVERPROP...	BATCH COMPLETED	sqladmin	Microsoft SQL Ser...

Figure 6.50 – KQL results of the audit data

How it works...

Please refer to the following to understand how auditing works in Azure SQL Database:

1. Auditing is not enabled by default. We enabled a database-level audit for the **adeawlt** database in Azure Portal.
2. Azure SQL Database auditing records login success, login failures, and batch completion events when configured using Azure portal. If you need additional events to be added to the audit policy, you may do so using the PowerShell command, **Set-AzSqlDatabaseAudit**.
3. We configured the audit to store the data to our **packtadesqllgw** Log Analytic workspace.
4. We can use a single Log Analytic workspace (**packtadesqllgw**) to store both the audit data and SQL diagnostics data. How long the Log Analytic workspace will retain the audit logs depends on the workspace's data retention settings.
5. KQL scripts are used to query the Log Analytic workspace. **AzureDiagnostics** is the single table that maintains both SQL audit data and database diagnostics. By filtering for SQL audit events using the `Category == 'SQLSecurityAuditEvents'` condition, we were able to find audit records. By filtering using the `action_name_s == "BATCH_COMPLETED"` condition, we were able to find queries or batch completion events and verify whether our actions on the database were recorded or not.

7

Processing Data Using Azure Databricks

Databricks is a data engineering product built on top of Apache Spark that provides a unified, cloud-optimized platform so that you can perform **Extract, Transform, and Load (ETL)**, **Machine Learning (ML)**, and **Artificial Intelligence (AI)** tasks on a large quantity of data.

Azure Databricks, as its name suggests, is the Databricks integration with Azure, which also provides fully managed Spark clusters, an interactive workspace for data visualization and exploration, integration with data sources such as Azure Blob Storage, Azure Data Lake Storage, Azure Cosmos DB, and Azure SQL Data Warehouse.

Azure Databricks can process data from multiple and diverse data sources, such as SQL or NoSQL, structured or unstructured data, and streaming data sources, and also scale up as many servers as required to cater to any data growth.

By the end of the chapter, you will have learned how to configure Databricks, work with storage accounts, process data using Scala, store processed data in Delta Lake, and visualize the data in Power BI.

In this chapter, we'll cover the following recipes:

- Configuring the Azure Databricks environment
- Integrate Databricks with Azure Key Vault
- Mounting an Azure Data Lake container in Databricks
- Processing data using notebooks
- Scheduling notebooks using job clusters
- Working with Delta Lake tables
- Connecting a Databricks Delta Lake to Power BI

Technical requirements

For this chapter, you will need the following:

- A Microsoft Azure subscription
- PowerShell 7
- Microsoft Azure PowerShell
- Power BI Desktop

Configuring the Azure Databricks environment

In this recipe, we'll learn how to configure the Azure Databricks environment by creating an Azure Databricks workspace, cluster, and cluster pools.

Getting ready

To get started, log in to <https://portal.azure.com> using your Azure credentials.

How to do it...

An Azure Databricks workspace is the starting point for writing solutions in Azure Databricks. A workspace is where you create clusters, write notebooks, schedule jobs, and manage the Azure Databricks environment.

An Azure Databricks workspace can be created in an Azure-managed virtual network or customer-managed virtual network. In this recipe, we will create a Databricks cluster in an Azure-managed network. Let's get started:

1. Go to portal.azure.com and click **Create a resource**. Search for **Azure Databricks**. Click **Create**, as shown in the following screenshot:

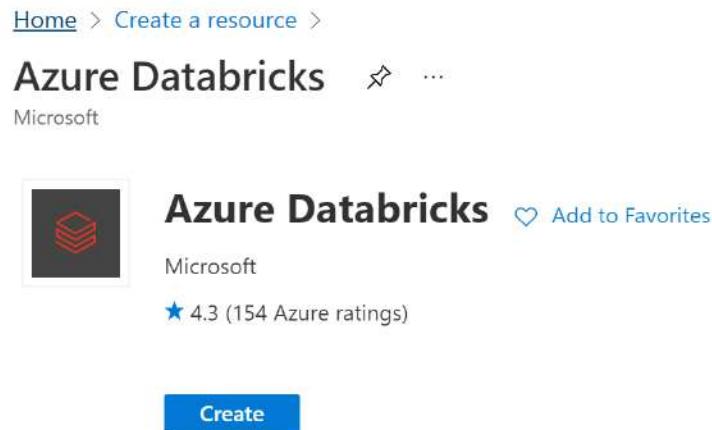


Figure 7.1 – Creating a Databricks resource

2. Provide the resource group name and workspace name, as shown in the following screenshot, and click **Review + Create**:

A screenshot of the 'Create an Azure Databricks workspace' configuration page. It shows the 'Basics' tab selected. The 'Project Details' section has a note about managing resources via a subscription and resource groups. The 'Subscription' dropdown is set to 'Visual Studio Enterprise Subscription'. The 'Resource group' dropdown is set to '(New) packtadedb', with a red box highlighting the dropdown and the 'Create new' link below it. The 'Instance Details' section includes fields for 'Workspace name' (set to 'pactadedabrics'), 'Region' (set to 'East US'), and 'Pricing Tier' (set to 'Standard (Apache Spark, Secure with Azure AD)').

Home > Create a resource > Azure Databricks >

Create an Azure Databricks workspace

Basics Networking Advanced Tags Review + create

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) packtadedb [Create new](#)

Instance Details

Workspace name * pactadedabrics

Region * East US

Pricing Tier * ⓘ Standard (Apache Spark, Secure with Azure AD)

Figure 7.2 – Creating a Databricks workspace

Creating Azure Databricks clusters

Once the resource is created, go to the Databricks workspace that we created (go to `portal.azure.com`, click on **All resources** and search for `pactadedatabricks`). Perform the following steps to create a Databricks cluster:

1. Click **Launch Workspace**:

 Delete

^ Essentials

Status : Active	Managed Resource Group : databricks-rg-pactadedatabricks-6qmidgjocusbo
Resource group : pactadedb	URL : https://adb-7675839323985314.14.azuredatabricks.net
Location : East US 	Pricing Tier : standard
Subscription : Visual Studio Enterprise Subscription	
Subscription ID :	
Tags (edit) : Click here to add tags	



Launch Workspace

Figure 7.3 – Launching the workspace

2. To create a cluster, select **Compute** from the left-hand menu of the Databricks workspace:

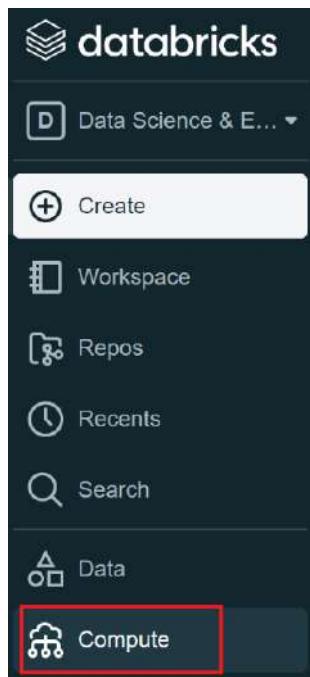


Figure 7.4 – Creating a cluster

There are two types of cluster: **Interactive** and **Automated**. Interactive clusters are created manually by users so that they can interactively analyze the data while working on, or even developing, a data engineering solution. Automated clusters are created automatically when a job starts and are terminated as and when the job completes.

3. Click **Create Cluster** to create a new cluster. On the **New Cluster** page, provide a cluster name of `dbcluster01`. Then, set **Cluster mode** to **Standard**, **Terminate after** to **10 minutes of inactivity**, **Min workers** to **1**, and **Max workers** to **2**, and leave the rest of the options as their defaults:

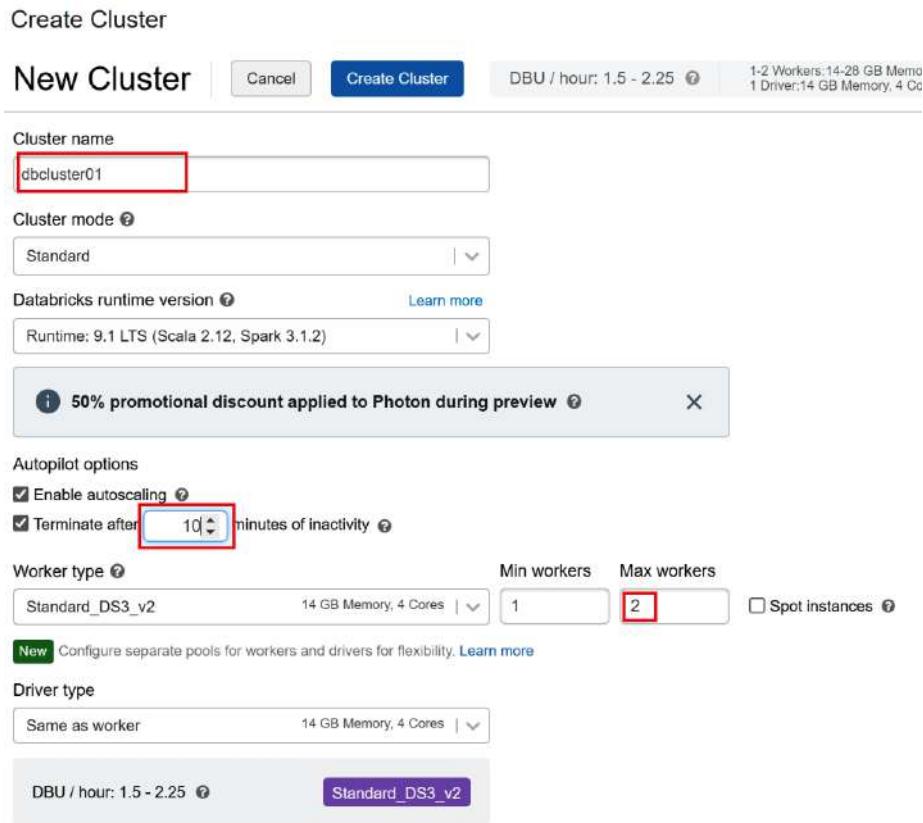


Figure 7.5 – Creating a cluster configuration

There are two major cluster modes: **Standard** and **High Concurrency**. **Standard** cluster mode uses single-user clusters, optimized to run tasks one at a time. The **High Concurrency** cluster mode is optimized to run multiple tasks in parallel; however, it only supports R, Python, and SQL workloads, and doesn't support Scala.

These autoscaling options allow Databricks to provision as many nodes as required to process a task within the limit, as specified by the **Min workers** and **Max workers** options.

The **Terminate after** option terminates the clusters when there's no activity for a given amount of time. In our case, the cluster will auto-terminate after 10 minutes of inactivity. This option helps save costs.

There are two types of cluster nodes: **Worker type** and **Driver type**. The **Driver type** node is responsible for maintaining a notebook's state information, interpreting the commands being run from a notebook or a library, and co-ordinates with Spark executors. The **Worker type** nodes are the Spark executor nodes, which are responsible for distributed data processing.

The **Advanced options** section can be used to configure Spark configuration parameters, environment variables, and tags, configure **Secure Shell (SSH)** in the clusters, enable logging, and run custom initialization scripts at the time of cluster creation.

4. Click **Create Cluster** to create the cluster. It will take around 5 to 10 minutes to create the cluster, and may take more time depending on the number of worker nodes that you have selected.

Creating Azure Databricks pools

Azure Databricks pools reduce cluster startup and autoscaling time by keeping a set of idle, ready-to-use instances without the need for creating instances when required. To create Azure Databricks pools, execute the following steps:

1. In your Azure Databricks workspace, on the **Compute** page, select the **Pools** option, and then select **Create Pool** to create a new pool. Provide the pool's name, then set **Min Idle** to 2, **Max Capacity** to 4, and **Idle Instance Auto Termination** to 10. Leave **Instance Type** as its default of **Standard_DS3_v2** and set **Preloaded Databricks Runtime Version** to **Runtime: 9.1 LTS (Scala 2.12, Spark 3.1.2)**:

Clusters / Pools / Create Pool

Create Pool

Name

Min Idle (2)

Max Capacity (4)

Idle Instance Auto Termination (10)

Terminate instances above minimum after minutes of idle time.

Instance Type (Standard_DS3_v2) 14 GB Memory, 4 Cores

Preloaded Databricks Runtime Version (Runtime: 9.1 LTS (Scala 2.12, Spark 3.1.2))

Instances Tags

On-demand/Spot

All On-demand All Spot

Figure 7.6 – Creating a Databricks cluster pool

Min Idle specifies the number of instances that will be kept idle and available without terminating. The **Idle Instance Auto Terminate** setting doesn't apply to these instances. **Max Capacity** limits the maximum number of instances to this number, including idle and running ones. This helps with managing cloud quotas and their costs.

The Azure Databricks runtime is a set of core components or software that runs on your clusters. There are different runtimes, depending on the type of workload you have.

2. Click **Create** to create the pool. We can attach a new or existing cluster to a pool by specifying the pool name under the **Worker type** option and **Driver type** option. In the workspace, navigate to the **Clusters** page and select **dbcluster01**, which we created in *step 2* of the previous section. On the **dbcluster01** page, click **Edit**, and select **dbclusterpool** from the **Worker type** drop-down list and **Driver type** drop-down list:

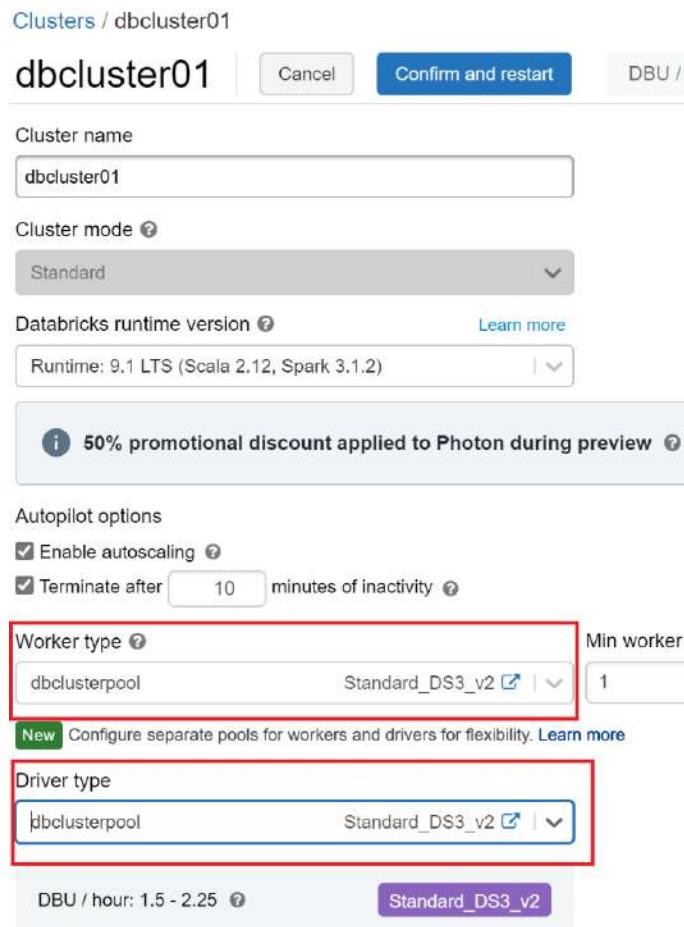


Figure 7.7 – Attaching a Databricks cluster to a pool

- Click **Confirm** to apply these changes. The cluster will now show up in the **Attached Clusters** list on the pool's page:

The screenshot shows the 'Clusters / Pools / Pool Details' interface for a pool named 'dbclusterpool'. The 'Overview' tab is selected. At the top, there are 'Edit' and 'Delete' buttons. Below the pool name, there are two sections: 'Instance Type: Standard_DS3_v2, 14 GB Memory, 4 Cores' with 'Min Idle: 2' and 'Idle Instance Auto Termination: 10 minutes', and 'Max Capacity: 4'. Under the 'Attached Clusters' section, a table lists one cluster: 'dbcluster01' (Name), 'Running' (State), and '2' (Nodes). A green checkmark is next to 'dbcluster01'.

Name	State	Nodes
dbcluster01	Running	2

Figure 7.8 – Databricks clusters attached to a pool

We can add multiple clusters to a pool. Whenever an instance, such as **dbcluster01**, requires an instance, it'll attempt to allocate the pool's idle instance. If an idle instance isn't available, the pool expands to get new instances, as long as the number of instances is under the maximum capacity.

Integrating Databricks with Azure Key Vault

Azure Key Vault is a useful service for storing keys and secrets that are used by various other services and applications. It is important to integrate Azure Key Vault with Databricks, as you could store the credentials of objects such as a SQL database or data lake inside the key vault. Once integrated, Databricks can reference the key vault, obtain the credentials, and access the database or data lake account. In this recipe, we will cover how you can integrate Databricks with Azure Key Vault.

Getting ready

Create a Databricks workspace and a cluster as explained in the *Configuring the Azure Databricks environment* recipe of this chapter.

Log in to `portal.azure.com`, click **Create a resource**, search for **Key Vault**, and click **Create**. Provide the key vault details, as shown in the following screenshot, and click **Review + create**:

Home > packtadedb > Create a resource > Key Vault >

Create a key vault

...

Basics Access policy Networking Tags Review + create

Azure Key Vault is a cloud service used to manage keys, secrets, and certificates. Key Vault eliminates the need for developers to store security information in their code. It allows you to centralize the storage of your application secrets which greatly reduces the chances that secrets may be leaked. Key Vault also allows you to securely store secrets and keys backed by Hardware Security Modules or HSMs. The HSMs used are Federal Information Processing Standards (FIPS) 140-2 Level 2 validated. In addition, key vault provides logs of all access and usage attempts of your secrets so you have a complete audit trail for compliance.

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 * packtadedb ▼

[Create new](#)

Instance details

Key vault name * packtadedbkv ✓

Region * East US ▼

Pricing tier * Standard ▼

Recovery options

Soft delete protection will automatically be enabled on this key vault. This feature allows you to recover or permanently delete a key vault and secrets for the duration of the retention period. This protection applies to the key vault and the secrets stored within the key vault.

To enforce a mandatory retention period and prevent the permanent deletion of key vaults or secrets prior to the retention period elapsing, you can turn on purge protection. When purge protection is enabled, secrets cannot be purged by users or by Microsoft.

[Review + create](#)

[< Previous](#)

[Next : Access policy >](#)

Figure 7.9 – Creating a key vault

How to do it...

Perform the following steps to integrate Databricks with Azure Key Vault:

1. Open the **packtadedatabricks** Databricks workspace on the Azure portal and click **Launch Workspace**.

2. This will open up the Databricks portal. Copy the URL, which will be set out as `https://adb-xxxxxxxxxxxxxx.xx.azuredatabricks.net/?o=xxxxxxxxxxxxxx#:`

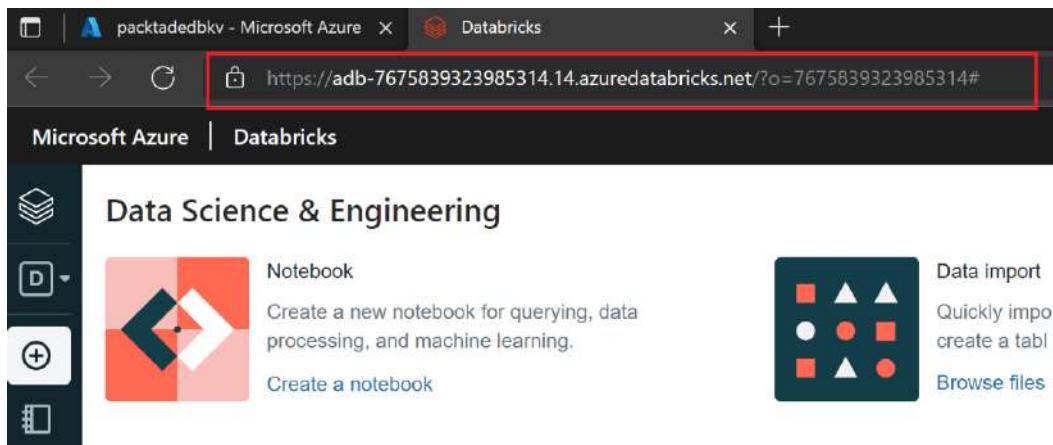


Figure 7.10 – The Databricks URL

3. Add `secrets/createScope` to the end of the URL and go to the URL `https://adb-xxxxxxxxxxxxxx.xx.azuredatabricks.net/?o=xxxxxxxxxxxxxx#secrets/createScope`. It should open a page to create a secret scope. Provide the following details:
 - Scope name:** Any name that relates to the key or password that you will access through the key vault. In our case, let's use `datalakekey`.
 - DNS Name:** DNS would be the DNS name of Key Vault, which will have the following format: `<key vault name>.vault.azure.net`. In our case, it is, `packtadedbkv.vault.azure.net`.
 - Set Manage Principal** to All Users.
 - Resource ID:** Resource ID of the key vault. To obtain it, go to **Key Vault** in the Azure portal (under **All resources**, search for `packtadedbkv`). Go to **Properties** and copy the **Resource ID**:

The screenshot shows the 'Properties' page for a Key Vault named 'packtadedbkv'. The 'Properties' tab is selected. The 'Resource ID' field contains a long GUID string: '/subscriptions/[REDACTED]/resourceGroups/packtade...'. A red box highlights this field, and another red box highlights the 'Copy to clipboard' button to its right.

Figure 7.11 – The key vault resource ID

4. Return to the Databricks portal secret scope page and fill in the details, as shown in the following screenshot:

The screenshot shows the 'Create Secret Scope' page in the Databricks portal. The 'Scope Name' field is set to 'datalakekey'. In the 'Azure Key Vault' section, the 'DNS Name' field is set to 'https://packtadedbkv.vault.azure.net/' and the 'Resource ID' field is set to the same long GUID string as in Figure 7.11. The 'Manage Principal' dropdown is set to 'All Users'. The 'Create' button is visible at the top right.

Figure 7.12 – Key vault scope creation

You will receive confirmation that a secret scope called **datalakekey** has been successfully added. This completes the integration between Databricks and Azure Key Vault.

How it works...

Upon completion of the preceding steps, all users with access to the Databricks workspace will be able to extract secrets and keys from the key vault and use them in a Databricks notebook to perform the desired operations.

Behind the scenes, Databricks uses a service principal to access the key vault. As we create the scope in the Databricks portal, Azure will grant the relevant permissions required for the Databricks service principal on the key vault. You can verify as much using the following steps:

1. Go to the **packtadedbkv** key vault on the Azure portal.
2. Click on **Access policies**. You will notice the Azure Databricks service principal being granted **Get** and **List** permissions on secrets. This will allow the Databricks workspace to read secrets from the key vault:

The screenshot shows the 'Access policies' page for the 'packtadedbkv' key vault in the Azure portal. The left sidebar shows navigation options like Overview, Activity log, Access control (IAM), Tags, Diagnose and solve problems, Events, Settings, Keys, Secrets, Certificates, and Access policies (which is selected and highlighted with a red box). The main area displays the 'Enable Access to:' section with three checkboxes for Azure VM deployment, Resource Manager template deployment, and Azure Disk Encryption volume encryption. Below this, the 'Permission model' is set to 'Vault access policy' (radio button selected). The 'Current Access Policies' table lists one entry: 'AzureDatabricks' under the 'APPLICATION' section. To the right of the table, there are dropdown menus for 'Key Permissions' (0 selected), 'Secret Permissions' (2 selected, with 'Get' and 'List' checked, also highlighted with a red box), and 'Certificate Permissions' (0 selected). A 'Select all' checkbox is available for each permission type.

Figure 7.13 – Verifying permissions

Mounting an Azure Data Lake container in Databricks

Accessing data from Azure Data Lake is one of the fundamental steps of performing data processing in Databricks. In this recipe, we will learn how to mount an Azure Data Lake container in Databricks using the Databricks service principal. We will use Azure Key Vault to store the Databricks service principal ID and the Databricks service principal secret that will be used to mount a data lake container in Databricks.

Getting ready

Create a Databricks workspace and a cluster, as explained in the *Configuring the Azure Databricks environment* recipe of this chapter.

Create a key vault in Azure and integrate it with Azure Databricks, as explained in the *Integrating Databricks with Azure Key Vault* recipe.

Create an Azure Data Lake account, as explained in the *Provisioning an Azure Storage account using the Azure portal* recipe of *Chapter 1, Creating and Managing Data in Azure Data Lake*.

Go to the Azure Data Lake Storage account created in the Azure portal (click **All resources**, then search for `packtadestoragev2`). Click **Containers**. Click **+ Container**:

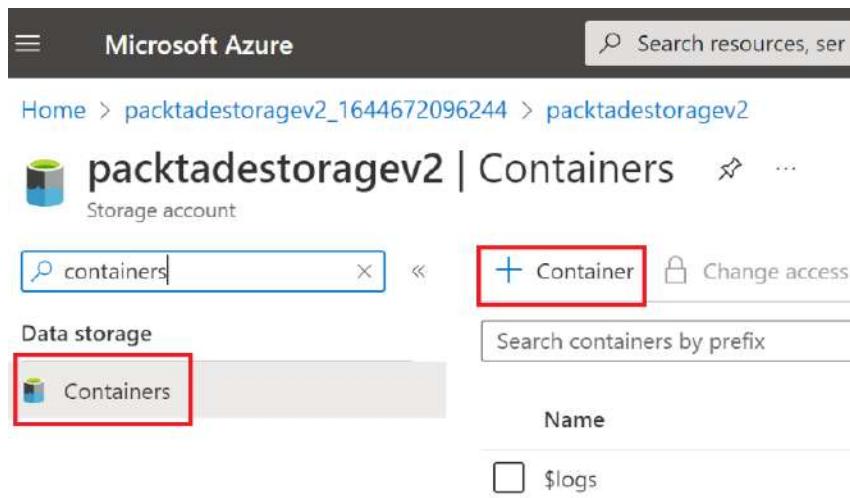


Figure 7.14 – Adding a container

Provide a container name of `databricks` and click **Create**:

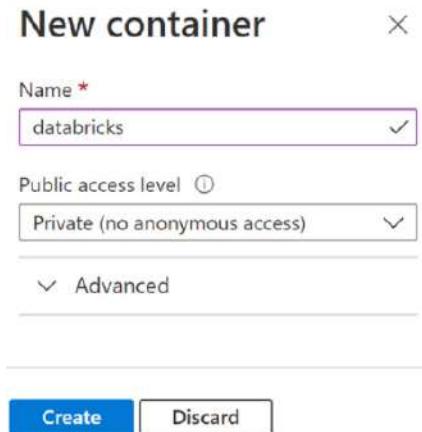


Figure 7.15 – Creating a container

How to do it...

Mounting the container in Databricks will involve the following high-level steps:

1. Registering an application in Azure **Active Directory (AD)** and obtaining the service principal secret from Azure AD
2. Storing the Application ID and service principal secret in a key vault
3. Granting permission on the data lake container to the Application ID
4. Mounting the data lake container on Azure Databricks

The detailed steps are as follows:

1. Go to portal.azure.com and click **Azure Active Directory**. Click **App registrations** and then hit **+ New registration**:

The screenshot shows the 'Default Directory | App registrations' page in the Azure Active Directory portal. At the top, there's a navigation bar with 'Home > Default Directory'. Below it is the main title 'Default Directory | App registrations' with a 'New registration' button highlighted by a red box. To the right are links for 'Endpoints', 'Troubleshooting', 'Refresh', and 'Download'. A message box informs users that starting June 30th, 2020, no new features will be added to Azure Active Directory, and applications will need to be upgraded to Microsoft Authentication. On the left, a sidebar titled 'Manage' lists various options: 'Overview', 'Preview features', 'Diagnose and solve problems', 'Users', 'Groups', 'External Identities', 'Roles and administrators', 'Administrative units', 'Enterprise applications', 'Devices', and 'App registrations', which is also highlighted by a red box. The main content area shows tabs for 'All applications', 'Owned applications' (which is underlined), 'Deleted applications', and 'Applications fr'. A search bar at the bottom allows filtering results.

Figure 7.16 – App registration

2. Provide a name of PacktDatabricks and click on the **Register** button:

Home > Default Directory >

Register an application

...

* Name

The user-facing display name for this application (this can be changed later).

PacktDatabricks



Supported account types

Who can use this application or access this API?

- Accounts in this organizational directory only (Default Directory only - Single tenant)
- Accounts in any organizational directory (Any Azure AD directory - Multitenant)
- Accounts in any organizational directory (Any Azure AD directory - Multitenant) and personal Microsoft accounts (e.g. Skype, Xbox)
- Personal Microsoft accounts only

[Help me choose...](#)

Redirect URI (optional)

We'll return the authentication response to this URI after successfully authenticating the user. Providing this now is optional and it can be changed later, but a value is required for most authentication scenarios.

Select a platform



e.g. https://example.com/auth

Register an app you're working on here. Integrate gallery apps and other apps from outside your organization by adding from [Enterprise applications](#).

By proceeding, you agree to the [Microsoft Platform Policies](#)

[Register](#)

Figure 7.17 – Registering an application

3. After the registration is done, copy the **Application (client) ID** and the **Directory (tenant) ID**:

Home > PacktDatabricks

Search (Ctrl+ /) Delete Endpoints Preview features

Overview

Essentials

Display name	: PacktDatabricks
Application (client) ID	: 5ef28dab-aa0d-4892-b427-97bcac7a8b
Object ID	: 5d3f096f-2cce-4fde-807f-00370be9ce56
Directory (tenant) ID	: [REDACTED]

Quickstart Integration assistant

Manage

Branding & properties

Figure 7.18 – Copying the Application ID

- Click **Certificates & secrets**. Click **+ New client secret**. For **Description**, provide any relevant description and click **Add**:

Microsoft Azure Search resources, services, and docs (G+) Home > Default Directory > PacktDatabricks

PacktDatabricks | Certificates & secrets

Search (Ctrl+ /) Got feedback?

Overview Quickstart Integration assistant

Manage

Branding & properties Authentication Certificates & secrets Token configuration API permissions Expose an API App roles Owners

Certificates (0) Client secrets (0) Federated credentials (0)

A secret string that the application uses to prove its identity when requesting a token. Also can be referred to as application pass.

+ New client secret

Description	Expires	Value	Secret
datalake	Recommended: 6 months	[REDACTED]	

Add Cancel

Figure 7.19 – Adding a client secret

- Copy the generated secret value. Ensure you copy it before closing the screen, as you only get to see it once:

Certificates (0) Client secrets (1) Federated credentials (0)			
A secret string that the application uses to prove its identity when requesting a token. Also can be referred to as application password.			
+ New client secret			
Description	Expires	Value ⓘ	Secret ID
datalake	8/12/2022	6zK7Q~chQjwZdOVQ7vdigvl2QDwgA11... ⓘ	a29a74c6-8d2c-4a52-8044-2fb9882558b4 ⓘ

Figure 7.20 – The secret value

6. Go to the **packtadedbkv** Azure Key Vault. Click **Secrets**. Click **+ Generate/Import**:

Home > packtadedbkv

packtadedbkv | Secrets

Key vault

secrets

+ Generate/Import

Settings

Secrets

Name

There are no secrets

Figure 7.21 – Creating secrets

7. Set **Name** as `appsecret` and **Value** as the secret value copied in *step 5*. Hit the **Create** button:

Home > packtadedbkv >

Create a secret

Upload options: Manual

Name * ⓘ appsecret

Value * ⓘ ⓘ

Content type (optional)

Set activation date ⓘ

Set expiration date ⓘ

Enabled: Yes ⚡ No

Tags: 0 tags

Create

Figure 7.22 – Storing secrets

8. Repeat *step 6* and *step 7*, and add the application ID and directory ID values saved in *step 3* as secrets inside the key vault. Provide the name for the secrets as **ApplicationID** and **DirectoryID**. Once done, the key vault should have three secrets, as shown in the following screenshot:

The screenshot shows the Azure Key Vault interface for the 'packtadedbkv' vault. On the left, a sidebar lists navigation options: Overview, Activity log, Access control (IAM), Tags, Diagnose and solve problems, Events, and Settings. The main area displays a message: 'The secret 'ApplicationID' has been successfully created.' Below this, a table lists three secrets:

Name	Type
ApplicationID	
appsecret	
DirectoryID	

Figure 7.23 – secrets added

9. Go to the **packtadestoragev2** Data Lake account. Click **Containers** and open the **databricks** container. Click **Access Control (IAM)** and then click **Add role assignment**:

The screenshot shows the 'databricks' container's Access Control (IAM) settings. The sidebar includes options: Overview, Diagnose and solve problems, and Access Control (IAM), which is highlighted with a red box. The main area features a 'My access' section with a 'View my access' button and a 'Check access' button. At the top, there are buttons for 'Add', 'Download role assignments', 'Edit column', and a 'Deny ass' button. A dropdown menu for 'Add role assignment' is open, showing options: 'Add co-administrator', 'Add role assignment' (which is also highlighted with a red box), and 'Deny ass'.

Figure 7.24 – Adding role assignment

10. Search for the Storage Blob Data Contributor role, select it, and click **Next**:

Add role assignment ...

Got feedback?

Role Members * Conditions (optional) Review + assign

A role definition is a collection of permissions. You can use the built-in roles or you can create your own custom roles. [Learn more](#) ⓘ

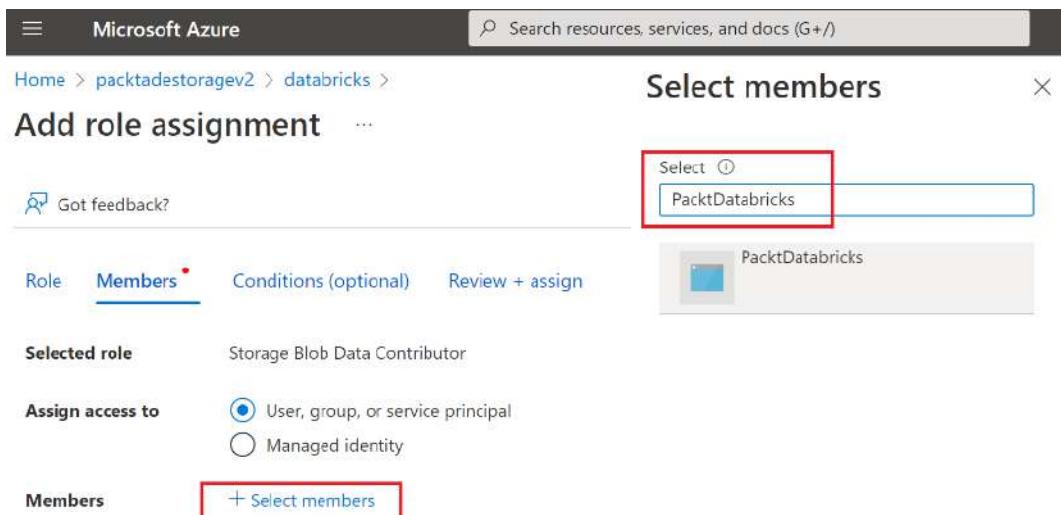
[Use classic experience](#) ⓘ



The screenshot shows the 'Add role assignment' interface. At the top, there are tabs for 'Role', 'Members *' (which is selected), 'Conditions (optional)', and 'Review + assign'. Below the tabs, a note says 'A role definition is a collection of permissions. You can use the built-in roles or you can create your own custom roles.' with a 'Learn more' link. A link to 'Use classic experience' is also present. The main area has a search bar with 'Storage Blob Data Contributor' typed in, which is highlighted with a red rectangle. To the right of the search bar are buttons for 'Type : All' and 'Category : All'. Below the search bar, it says 'Showing 1 of 31 roles'. A single result is listed: 'Storage Blob Data Contributor' with a description 'Allows for read, write and delete access to Azure Storage blob containers and data'. At the bottom, there are buttons for 'Review + assign', 'Previous', and 'Next'.

Figure 7.25 – Storage Blob Data Contributor

11. Click + **Select members** and, in the **Select members** box, type the app name registered in step 2, which is **PacktDatabricks**:



The screenshot shows the 'Select members' dialog box from the Microsoft Azure portal. The title is 'Select members' and there is a close button 'X'. On the left, there is a 'Select' button with a help icon and a list containing 'PacktDatabricks', which is highlighted with a red rectangle. Below the list is a thumbnail for 'PacktDatabricks'. At the bottom of the dialog, there is a 'Done' button.

The main page behind the dialog shows the 'Add role assignment' step. It has tabs for 'Role', 'Members *' (selected), 'Conditions (optional)', and 'Review + assign'. The 'Selected role' is 'Storage Blob Data Contributor'. Under 'Assign access to', the radio button for 'User, group, or service principal' is selected. In the 'Members' section, there is a button '+ Select members' which is highlighted with a red rectangle.

Figure 7.26 – Adding members to the Storage Blob Data Contributor role

12. Select **PacktDatabricks**, hit the **Select** button, and then click **Review + assign** to assign permission:

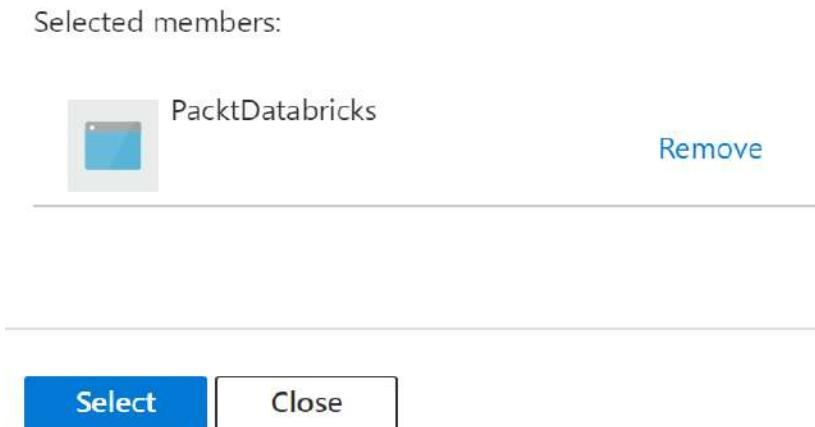


Figure 7.27 – Selecting members for the Storage Blob Data Contributor role

13. Launch the Databricks workspace if you need to and go back to the Databricks portal. From the **Create** menu, select **Notebook**:

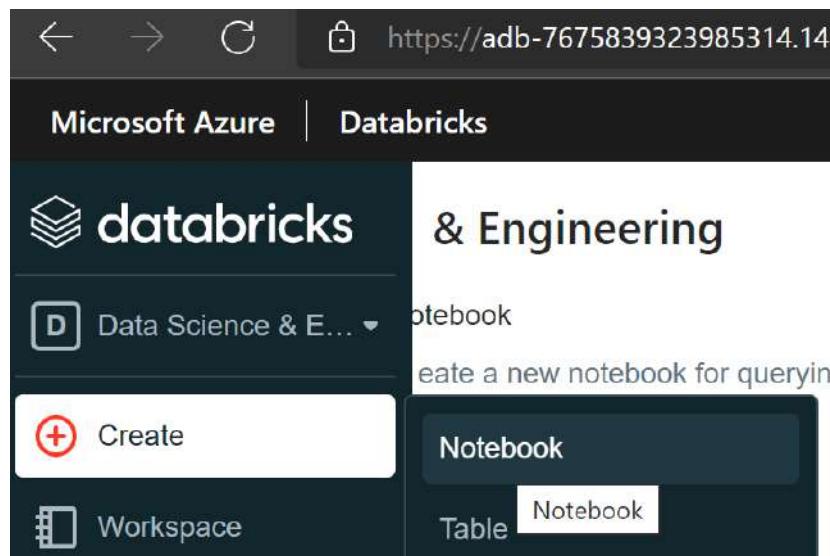


Figure 7.28 – Creating a notebook

14. Provide any notebook name. Set **Default Language** to **Scala**:

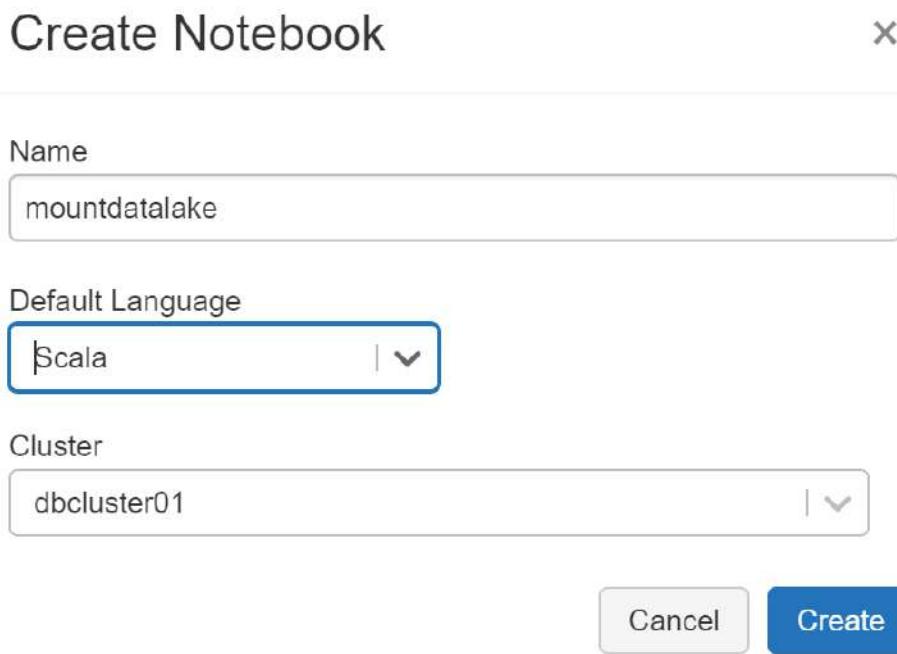


Figure 7.29 – The notebook name

15. Use the following Scala code to mount the data lake container in Databricks. The code extracts the application ID, application secret, and directory ID from the key vault using the `dbutils.secrets.get` function available in Scala. The `dbutils.secret.get` function takes the scope name (provided in the *Integrating Databricks with Azure Key Vault* recipe), and the secret names provided in *step 7* and *step 8*. The `dbutils.fs.mount` command has a parameter called `source`, which takes the URL of the data lake container to be mounted. The data lake container URL format is `abfss://<containernamespace>@<storageaccountname>.dfs.core.windows.net/` and, in our case, the URL would be `abfss://databricks@packtadestoragev2.dfs.core.windows.net/`:

```
val appsecret = dbutils.secrets.  
get(scope="datalakekey",key="appsecret")  
val ApplicationID = dbutils.secrets.  
get(scope="datalakekey",key="ApplicationID")  
val DirectoryID = dbutils.secrets.  
get(scope="datalakekey",key="DirectoryID")  
val endpoint = "https://login.microsoftonline.com/" +  
DirectoryID + "/oauth2/token"
```

```

val configs = Map(
    "fs.azure.account.auth.type" -> "OAuth",
    "fs.azure.account.oauth.provider.type" -> "org.apache.hadoop.fs.azurebfs.oauth2.ClientCredsTokenProvider",
    "fs.azure.account.oauth2.client.id" -> ApplicationID,
    "fs.azure.account.oauth2.client.secret" -> appsecret,
    "fs.azure.account.oauth2.client.endpoint" -> endpoint)
// Optionally, you can add <directory-name> to the source
// URI of your mount point.
dbutils.fs.mount(
    source = "abfss://databricks@packtadestoragev2.dfs.core.windows.net/",
    mountPoint = "/mnt/datalakestorage",
    extraConfigs = configs)

```

Upon running the script, the data lake container will be successfully mounted:

The screenshot shows a Databricks notebook interface with the following details:

- Title:** mountdatalake
- Scope:** Scala
- Run Type:** Scala
- Run Status:** Success
- Run ID:** dbcluster01
- Run Time:** 28.32 seconds
- Run Date:** 13/02/2022, 08:09:18
- Run Location:** dbcluster01
- Script Content (highlighted in red boxes):**

```

1 val appsecret = dbutils.secrets.get(scope="datalakekey",key="appsecret")
2 val ApplicationID = dbutils.secrets.get(scope="datalakekey",key="ApplicationID")
3 val DirectoryID = dbutils.secrets.get(scope="datalakekey",key="DirectoryID")
4 val endpoint = "https://login.microsoftonline.com/" + DirectoryID + "/oauth2/token"
5 val configs = Map(
6     "fs.azure.account.auth.type" -> "OAuth",
7     "fs.azure.account.oauth.provider.type" -> "org.apache.hadoop.fs.azurebfs.oauth2.ClientCredsTokenProvider",
8     "fs.azure.account.oauth2.client.id" -> ApplicationID,
9     "fs.azure.account.oauth2.client.secret" -> appsecret,
10    "fs.azure.account.oauth2.client.endpoint" -> endpoint)
11 // Optionally, you can add <directory-name> to the source URI of your mount point.
12 dbutils.fs.mount(
13     source = "abfss://databricks@packtadestoragev2.dfs.core.windows.net/",
14     mountPoint = "/mnt/datalakestorage",
15     extraConfigs = configs)

```
- Output:**

```

* (1) Spark Jobs
appsecret: String = [REDACTED]
ApplicationID: String = [REDACTED]
DirectoryID: String = [REDACTED]
endpoint: String = https://login.microsoftonline.com/[REDACTED]/oauth2/token
configs: scala.collection.immutable.Map[String,String] = Map(fs.azure.account.oauth2.client.secret -> [REDACTED], fs.azure.account.auth.type -> OAuth, fs.azure.account.oauth2.client.endpoint -> https://login.microsoftonline.com/[REDACTED]/oauth2/token, fs.azure.account.oauth.provider.type -> org.apache.hadoop.fs.azurebfs.oauth2.ClientCredsTokenProvider, fs.azure.account.oauth2.client.id -> [REDACTED])
res2: Boolean = true

```

Figure 7.30 – Mounting the data lake container notebook

How it works...

On Azure AD, we registered an application that created a service principal. Service principals are entities that applications can use to authenticate themselves to Azure services. We provided permissions for the application ID on the container to be accessed, which grants permission to the service principal created. We stored the credentials of the service principal (the application ID and secret) in Azure Key

Vault to ensure secure access to credentials. Databricks obtains the service principal credentials (the application ID and secret) from the key vault and uses the security context of the service principal to access the Azure Data Lake account. Databricks, while mounting the data lake account, retrieves the application ID, directory ID, and secret from the key vault and uses the service principal context to access the Azure Data Lake account. This process makes for a very secure method of accessing a data lake account for the following reasons:

- Sensitive information such as an application secret or directory ID is accessed programmatically and not used in plain text inside the notebook. This ensures sensitive information is not exposed to anyone who accesses the notebook.
- Developers who access the notebook needn't know the password or key of the data lake account. Using a key vault ensures that they can continue their development work, even without having direct access to the data lake account or database.
- Data lake accounts can be accessed through account keys too. However, we used service principals for authentication, as using service principals restricts access to the data lake account via applications, while accessing them using an account key or **Shared Access Signature (SAS)** token would provide direct login access to the data lake account via tools such as Azure Storage Explorer.

Processing data using notebooks

Databricks notebooks are the fundamental component in Databricks for performing data processing tasks. In this recipe, we will perform operations such as reading, filtering, cleaning a **Comma-Separated Value (CSV)** file, and gaining insights from it using a Databricks notebook written in Scala code.

Getting ready

Create a Databricks workspace and a cluster, as explained in the *Configuring the Azure Databricks environment* recipe.

Download the `covid-data.csv` file from the path at <https://github.com/PacktPublishing/Azure-Data-Engineering-Cookbook-2nd-edition/blob/main/chapter07/covid-data.csv>.

How to do it...

Let's process some data using Scala in a Databricks notebook by following the steps provided here:

1. Log in to `portal.azure.com`. Go to **All resources** and find **pactadedatabricks**, the Databricks workspace created in the *Configuring the Azure Databricks environment* recipe. Click **Launch Workspace** to log in to the Databricks portal.

2. From the **Create** menu, select **Notebook**:

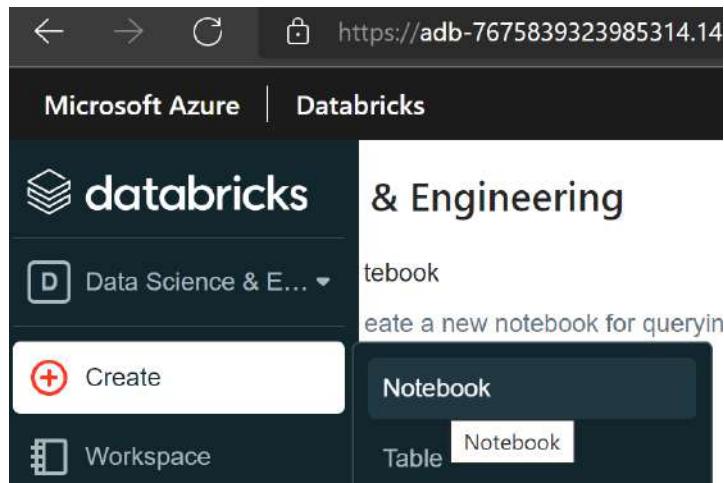


Figure 7.31 – Creating a notebook

3. Set the notebook name as `processdata` and **Default Language** to **Scala**:

A screenshot of a "Create Notebook" dialog box. At the top, it says "Create Notebook" and has a close button "x". Below that, there's a "Name" field containing "processdata". Under "Default Language", there's a dropdown menu set to "Scala". In the "Cluster" section, there's a dropdown menu set to "dbcluster01". At the bottom right, there are two buttons: "Cancel" and "Create", with "Create" being the primary button.

Figure 7.32 – Creating the `processdata` notebook

4. In the **File** menu, click **Upload Data**:

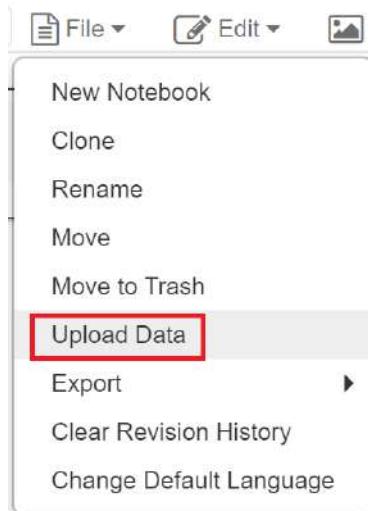


Figure 7.33 – Uploading data

5. Click **Drop files to upload, or click to browse**. Make sure to also note down the default path in your environment. Usually, it will follow the following format – /FileStore/shared_uploads/<loginname>:

Upload Data

DBFS Target Directory [?](#)

/FileStore/ shared_uploads/arr.nagaraj@gmail.com [Select](#)

Files [?](#)

Drop files to upload, or click to browse

A screenshot of a 'Upload Data' interface. It shows a 'DBFS Target Directory' field containing the path '/FileStore/ shared_uploads/arr.nagaraj@gmail.com'. A 'Select' button is to the right of the field. Below it, a message states 'Files uploaded to DBFS are accessible by everyone who has access to this workspace. [Learn more](#)'. Underneath, a section titled 'Files' shows a placeholder text 'Drop files to upload, or click to browse'.

Figure 7.34 – Uploading data

6. Upload the covid-data.csv file downloaded at the beginning of this recipe:

Upload Data

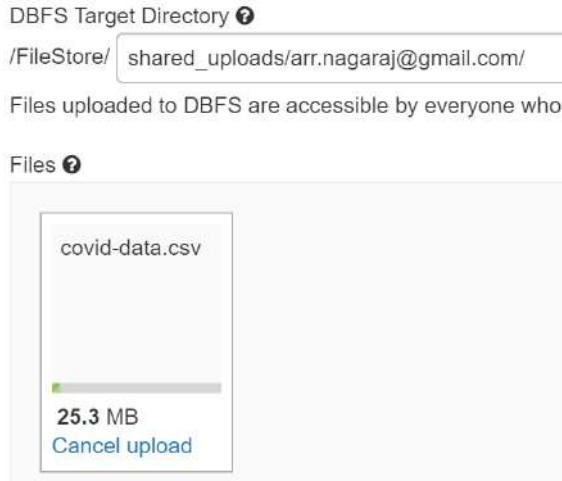


Figure 7.35 – Uploading the covid-data.csv file

7. Execute the following command to read the data to a DataFrame. Ensure to replace the file path noted in *step 6*. Notice from the output message that the DataFrame contains over 60 fields:

```
val covid_raw_data = spark.read.format("csv")
.option("header", "true")
.option("inferSchema", "true")
.load("/FileStore/shared_uploads/arr.nagaraj@gmail.com/
covid_data.csv")
```

The result of the command is provided here:

```
val covid_raw_data = spark.read.format("csv")
.option("header", "true")
.option("inferSchema", "true")
.load("/FileStore/shared_uploads/arr.nagaraj@gmail.com/covid_data.csv")

▶ (2) Spark Jobs
▶ covid_raw_data: org.apache.spark.sql.DataFrame = [iso_code: string, continent: string ... 59 more fields]
covid_raw_data: org.apache.spark.sql.DataFrame = [iso_code: string, continent: string ... 59 more fields]
Command took 10.04 seconds -- by arr.nagaraj@gmail.com at 13/02/2022, 23:07:48 on dbcluster01
```

Figure 7.36 – Reading the CSV data

8. At the right-hand corner of the cell, click on the drop-down button, and click **Add Cell Below**. Provide the following command to display the DataFrame:

```
display(covid_raw_data)
```

The result of the command is provided here:

	iso_code	continent	location	date	total_cases	new_cases	new_cases_smoothed	total_deaths	new_deaths	new_deaths_smoothed	total_cases_per_million	new_
1	AFG	Asia	Afghanistan	1/1/2021	52513	183	131.143	2201	12	9.429	1318.249	4.594
2	AFG	Asia	Afghanistan	2/1/2021	52586	73	117.429	2211	10	9	1320.081	1.835
3	AFG	Asia	Afghanistan	3/1/2021	52709	123	123	2221	10	9	1323.169	3.086
4	AFG	Asia	Afghanistan	4/1/2021	52909	200	128.857	2230	9	8.571	1328.19	5.021
5	AFG	Asia	Afghanistan	5/1/2021	53011	102	123.429	2237	7	7.857	1330.75	2.561
6	AFG	Asia	Afghanistan	6/1/2021	53105	94	110.714	2244	7	7.857	1333.11	2.36
7	AFG	Asia	Afghanistan	7/1/2021	53207	102	125.286	2253	9	9.143	1335.67	2.561

Figure 7.37 – Displaying the CSV data

9. Execute the following command to get the row count in the CSV file or DataFrame. There are over 94,000 rows:

```
covid_raw_data.count()
```

The result of the command is provided here:

```
Cmd 3
```

```
covid_raw_data.count()
```

▶ (2) Spark Jobs

```
res6: Long = 94342
```

Command took 1.74 seconds -- by arr.nagaraj@gmail.com at 13/02/2022, 12:22:56 on dbcluster01

Figure 7.38 – Displaying the row count

10. Let's remove any duplicates. The `dropDuplicates()` function removes duplicates and stores the result in a new DataFrame:

```
val covid_remove_duplicates = covid_raw_data.  
  dropDuplicates()
```

The result of the command is provided here:

```
Cmd 4  
  
val covid_remove_duplicates = covid_raw_data.dropDuplicates()  
  
▶ covid_remove_duplicates: org.apache.spark.sql.Dataset[org.apache.spark.sql.Row] = [iso_code: string, continent: string ... 59 more fields]  
covid_remove_duplicates: org.apache.spark.sql.Dataset[org.apache.spark.sql.Row] = [iso_code: string, continent  
Command took 0.41 seconds -- by arr.nagaraj@gmail.com at 13/02/2022, 12:24:32 on dbcluster01
```

Figure 7.39 – Removing duplicates

11. Let's look at the fields and their data type in the DataFrame. The `printSchema()` function provides the DataFrame structure:

```
covid_remove_duplicates.printSchema()
```

The result of the preceding command is provided in the following screenshot:

```
Cmd 5  
  
covid_remove_duplicates.printSchema()  
  
root  
|-- iso_code: string (nullable = true)  
|-- continent: string (nullable = true)  
|-- location: string (nullable = true)  
|-- date: string (nullable = true)  
|-- total_cases: integer (nullable = true)  
|-- new_cases: integer (nullable = true)  
|-- new_cases_smoothed: double (nullable = true)  
|-- total_deaths: integer (nullable = true)  
|-- new_deaths: integer (nullable = true)  
|-- new_deaths_smoothed: double (nullable = true)  
|-- total_cases_per_million: double (nullable = true)  
|-- new_cases_per_million: double (nullable = true)  
|-- new_cases_smoothed_per_million: double (nullable = true)  
|-- total_deaths_per_million: double (nullable = true)  
|-- new_deaths_per_million: double (nullable = true)  
|-- new_deaths_smoothed_per_million: double (nullable = true)  
|-- reproduction_rate: double (nullable = true)  
|-- icu_patients: integer (nullable = true)  
|-- icu_patients_per_million: double (nullable = true)  
|-- hosp_patients: integer (nullable = true)  
Command took 0.21 seconds -- by arr.nagaraj@gmail.com at 13/02/2022, 12:24:45 on dbcluster01
```

Figure 7.40 – Removing duplicates

12. The data is about the impact of COVID across the globe. Let's focus on a handful of columns that are required, instead of all the columns provided. The `select` function can help us to specify the columns that we need out of a DataFrame. Execute the following command to load the selected columns to another DataFrame:

```
val covid_selected_columns = covid_remove_duplicates.  
select("iso_code", "location", "continent", "date", "new_  
deaths_per_million", "people_fully_  
vaccinated", "population")
```

The result of the preceding command is provided in the following screenshot:

```
Cmd 6  
  
val covid_selected_columns = covid_remove_duplicates.select("iso_code", "location", "continent", "date", "new_deaths_per_million",  
"people_fully_vaccinated", "population")  
  
‣ [1] covid_selected_columns: org.apache.spark.sql.DataFrame = [iso_code: string, location: string ... 5 more fields]  
covid_selected_columns: org.apache.spark.sql.DataFrame = [iso_code: string, location: string ... 5 more fields]  
Command took 0.32 seconds -- by arr.nagaraj@gmail.com at 13/02/2022, 23:47:57 on dbcluster01
```

Figure 7.41 – Loading selected columns

13. For our analysis, let's remove any rows that contain NULL values in any of the columns. The `na.drop()` function can help to achieve this. Execute the following command. The `covid_clean_data` DataFrame will only contain rows without any NULL value in them once the command has been executed. The `covid_clean_data.count()` command shows that only 32,000+ rows were without any NULL values:

```
val covid_clean_data = covid_selected_columns.na.drop()  
covid_clean_data.count()
```

The result of the preceding command is provided in the following screenshot:

```
Cmd 7  
  
val covid_clean_data = covid_selected_columns.na.drop()  
covid_clean_data.count()  
  
‣ (3) Spark Jobs  
‣ [1] covid_clean_data: org.apache.spark.sql.DataFrame = [iso_code: string, location: string ... 5 more fields]  
covid_clean_data: org.apache.spark.sql.DataFrame = [iso_code: string, location: string ... 5 more fields]  
res7: Long = 32607  
Command took 3.45 seconds -- by arr.nagaraj@gmail.com at 13/02/2022, 23:49:28 on dbcluster01
```

Figure 7.42 – Removing NULL values

14. Data analysis is easier to perform using Spark SQL commands. To use SQL commands to analyze a DataFrame, we need to create a temporary view. The following command will create a temporary view called covid_view:

```
covid_clean_data.createOrReplaceTempView("covid_view")
```

The result of the preceding command is provided in the following screenshot:

Cmd 8

```
covid_clean_data.createOrReplaceTempView("covid_view")
```

Command took 0.17 seconds -- by arr.nagaraj@gmail.com at 13/02/2022, 23:55:46 on dbcluster01

Figure 7.43 – Creating a temporary view

15. Execute the following command to get some insights out of the data. %sql, on the first line, lets us switch from **Scala** to **SQL**. The SQL query provides the number of deaths, and the percentage of people vaccinated in each country with a population of over 1 million, ordered by countries with the highest deaths per million people:

```
%sql
SELECT iso_code, location, continent,
SUM(new_deaths_per_million) as death_sum,
MAX(people_fully_vaccinated * 100 / population) as
percentage_vaccinated FROM covid_view
WHERE population > 1000000
GROUP BY iso_code,location,continent
ORDER BY death_sum desc
```

The result of the command is provided here:

The screenshot shows a Jupyter Notebook interface. At the top, there's a command cell labeled 'Cmd 9' containing an SQL query:

```
%sql
Select iso_code, location , continent,
sum(new_deaths_per_million) as death_sum,
max(people_fully_vaccinated * 100 / population) as percentage_vaccinated From covid_view
where population > 1000000
group by iso_code,location,continent
order by death_sum desc
```

Below the command cell, it says '(3) Spark Jobs'. Under 'Table Data Profile', there is a table output:

	iso_code	location	continent	death_sum	percentage_vaccinated
1	BGR	Bulgaria	Europe	3705.9039999999986	29.377473572333255
2	PER	Peru	South America	2981.157	69.02865952535439
3	HUN	Hungary	Europe	2917.020999999997	63.64187149852784
4	CZE	Czechia	Europe	2436.6340000000005	63.598277709103584
5	TTO	Trinidad and Tobago	North America	2348.6360000000004	49.6739999458448
6	LVA	Latvia	Europe	2331.1030000000001	69.26527665145099
7	ROU	Romania	Europe	2301.099	41.90735857788351

Below the table, it says 'Showing all 156 rows.' and has icons for grid, chart, and download. The command took 2.63 seconds.

Figure 7.44 – Insights using the SQL query

16. To visualize the output from the previous SQL query, click on the chart icon and select Bar:

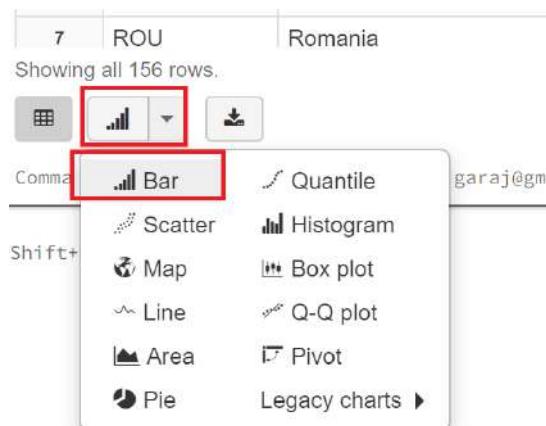


Figure 7.45 – Generating a bar graph

17. Click on Plot Options...:



Command took 1.69 seconds -- by arr.nagaraj@gmail.com

Figure 7.46 – Plot options

18. Set **Plot Options...** as follows. Set **location** in the **Keys** section, **continent** in the **Series groupings** section, and **death_sum** in the **Values** section, as shown in the following screenshot. This will provide a bar graph with a line for each country, with the color of the line based on the continent that the country belongs to:

Customize Plot

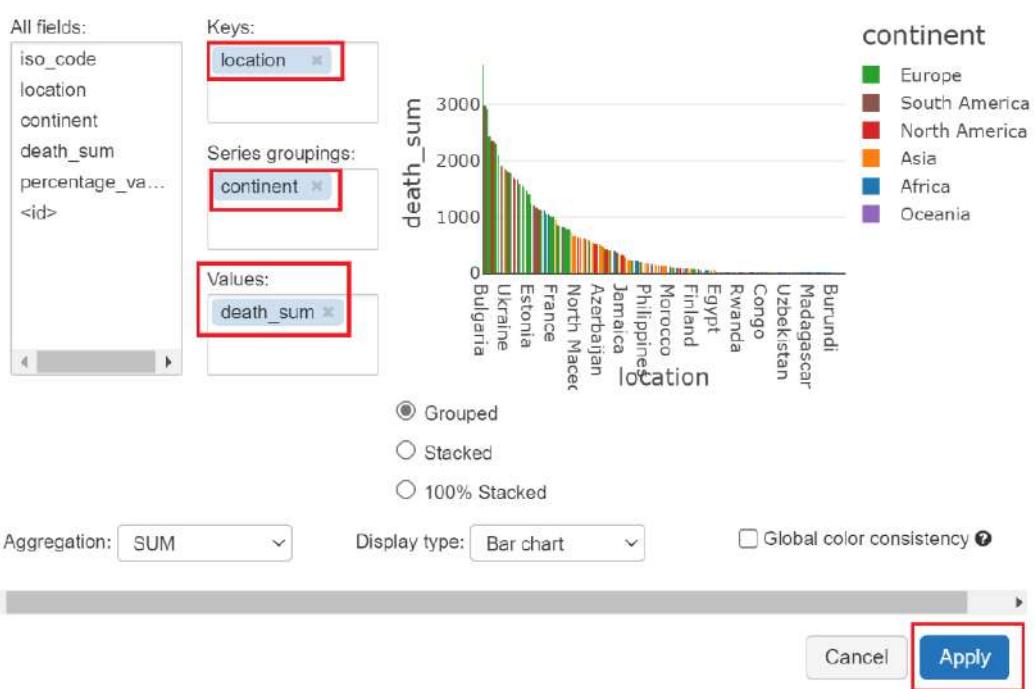


Figure 7.47 – Customizing the plot

19. You will get the following output. High spikes with green lines indicate that European countries were heavily affected by COVID:

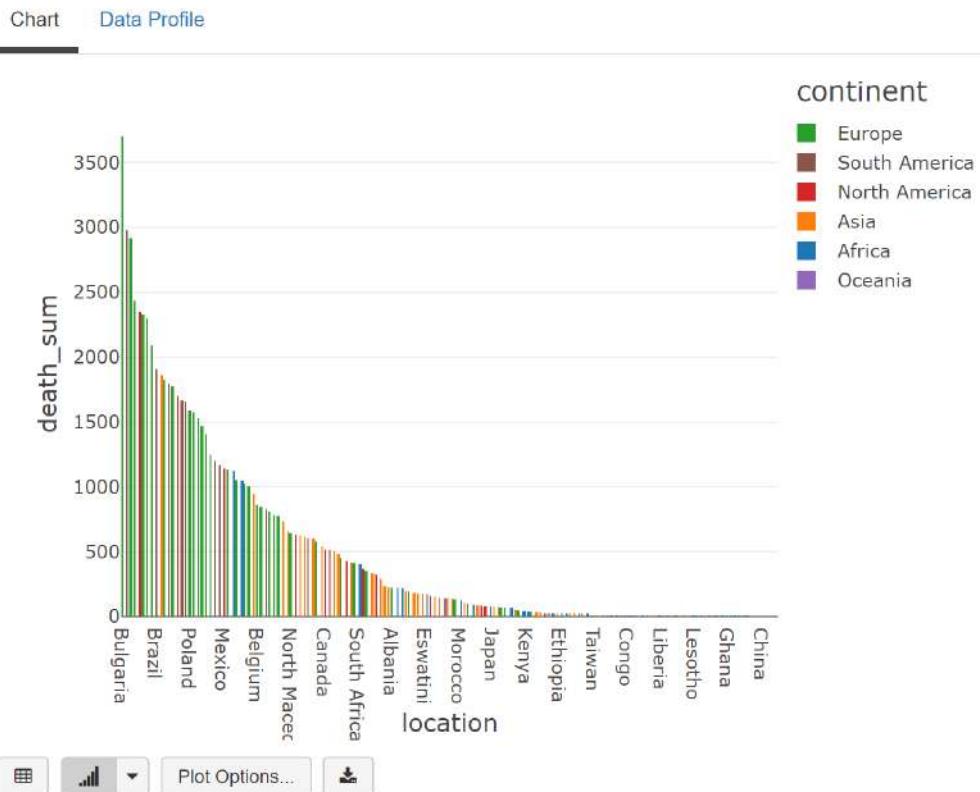


Figure 7.48 – Visual insights

How it works...

DataFrames are the fundamental objects used to store runtime data during data processing in Databricks. DataFrames are in-memory objects and extremely well-optimized for performing advanced analytics operations.

A CSV file was loaded to the **Databricks File System (DBFS)** storage, which is the default local storage available when a Databricks workspace is created. We can perform the same activities in a data lake account too, by uploading the CSV file to the data lake container and mounting the data lake container, as explained in the *Mounting an Azure Data Lake container in Databricks* recipe.

After loading the data to a DataFrame, we were able to cleanse the data by performing operations such as removing unwanted columns, dropping duplicates, and deleting rows with NULL values easily using Spark functions. Finally, by creating a temporary view out of the DataFrame, we were able to analyze the DataFrame's data using SQL queries and get visual insights using Databricks' visualization capabilities.

Scheduling notebooks using job clusters

Data processing can be performed using notebooks, but to operationalize it, we need to execute it at a specific scheduled time, depending upon the demands of the use case or problem statement. After a notebook has been created, you can schedule a notebook to be executed at a preferred frequency using job clusters. This recipe will demonstrate how you could schedule a notebook using job clusters.

Getting ready

Create a Databricks workspace, as explained in the *Configuring the Azure Databricks environment* recipe.

How to do it...

In the following steps, we will import the `SampleJob.dbc` notebook file into the Databricks workspace and schedule it to be run daily:

1. Log in to `portal.azure.com`. Go to **All resources** and find **pactadedatabricks**, the Databricks workspace created in the *Configuring the Azure Databricks environment* recipe. Click **Launch Workspace** to log in to the Databricks portal.
2. Navigate to **Workspace | Create | Folder**, as shown in the following screenshot:

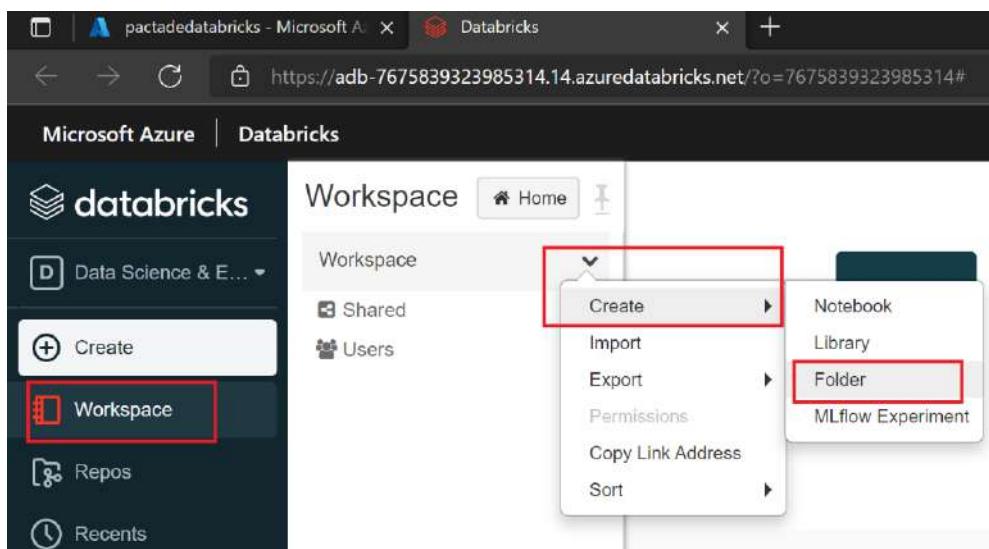


Figure 7.49 – Creating folder insights

3. Create a folder called `Job`:

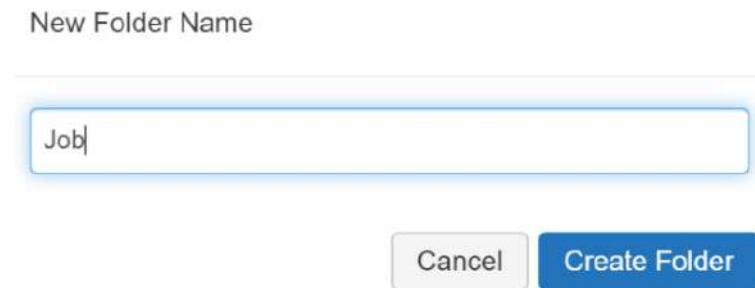


Figure 7.50 – Create a Job folder

4. Click **Workspace**, click on the **Job** folder, and pick the **Import** option:

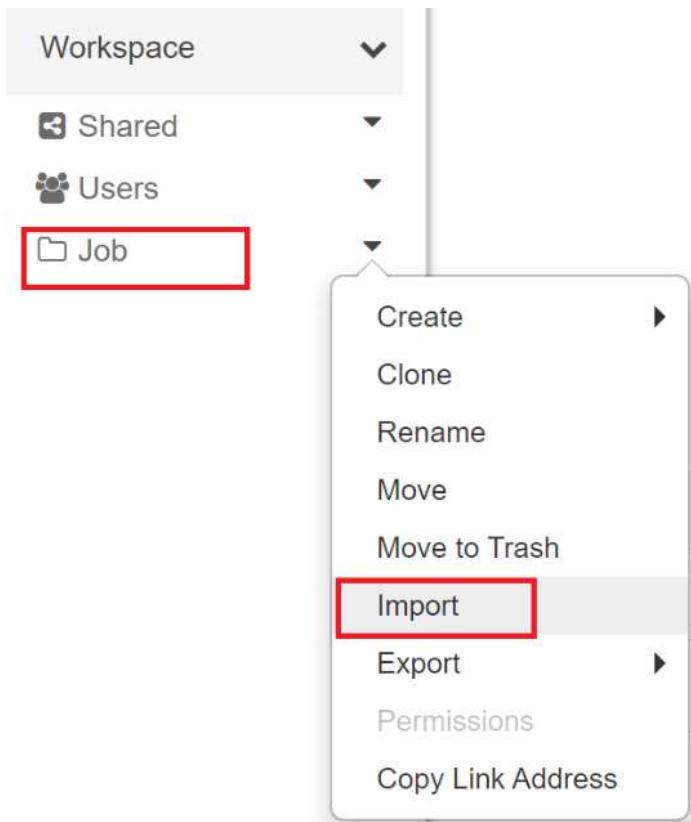


Figure 7.51 – Importing a notebook

- Pick the **URL** option to import the notebook. Paste the <https://github.com/PacktPublishing/Azure-Data-Engineering-Cookbook-2nd-edition/blob/main/chapter07/SampleJobdbc> path to import a notebook called **SampleJob**:

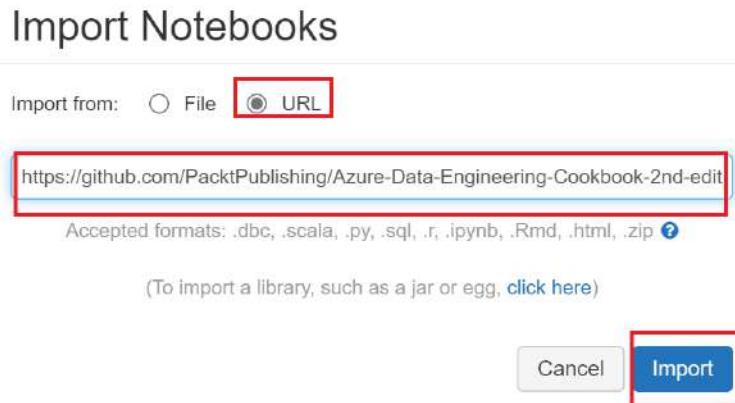


Figure 7.52 – Importing a notebook

- In the **Job** folder, you will have a notebook called **SampleJob**. The **SampleJob** notebook will read a sample CSV file and provide insights from it:

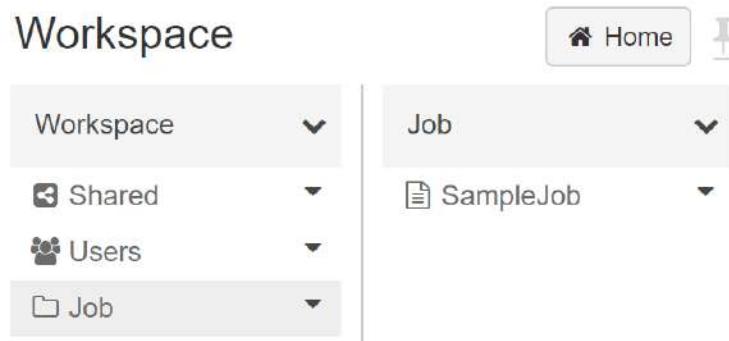


Figure 7.53 – The imported notebook

- From the menu on the left, click **Jobs**:

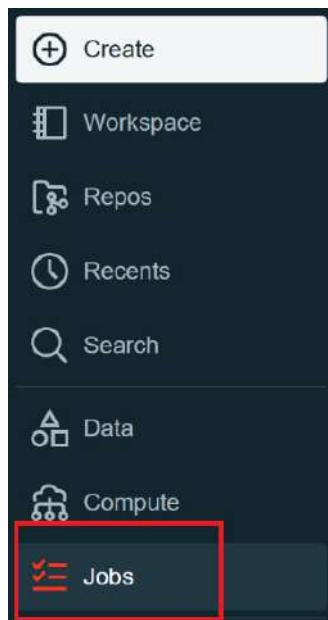


Figure 7.54 – Jobs in the Create menu

8. Click **Create Job**:

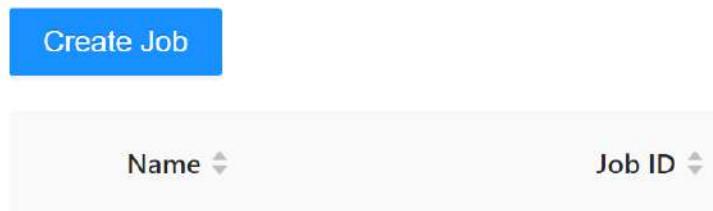


Figure 7.55 – Create Job

9. Provide a job name of **SampleJob**. Select the imported **SampleJob** notebook. On the **New Cluster** configuration, click on the edit icon to set the configuration options. The job will create a cluster each time it runs based on the configuration and delete the cluster once the job is completed:

Task name * ⓘ

Type * ⓘ
 | ⏺

Cluster * ⓘ

Parameters ⓘ [Add](#) [UI](#) | [JSON](#)

[Advanced options >](#)

Figure 7.56 – Editing the cluster configuration

10. Reduce the total number of cores to 2 and hit the **Confirm** button to save the cluster configuration.
 Hit the **Create** button on the job creation screen:

Worker type ⓘ
 14 GB Memory, 4 Cores | ⏺ Workers Spot instances ⓘ

New Configure separate pools for workers and drivers for flexibility. [Learn more](#)

Driver type
 14 GB Memory, 4 Cores | ⏺

DBU / hour: 2.25 ⓘ Standard_DS3_v2

▶ Advanced options

[Cancel](#) [Confirm](#)

Figure 7.57 – Save the cluster configuration

11. Hit the **Edit schedule** button to edit the schedule:

The screenshot shows the 'Job details' page for a notebook. At the top right are 'More ...' and 'Run now' buttons. Below them is a section titled 'Job details' with fields: 'Job ID' (89), 'Creator' (arr.nagaraj@gmail.com), and 'Run as' (arr.nagaraj@gmail.com). A section titled 'Schedule' shows 'None' and a red-bordered 'Edit schedule' button. Below that is a 'Clusters' section for 'SampleJob' with driver and worker details, and 'Configure' and 'Swap' buttons.

Figure 7.58 – Edit schedule configuration

12. Set **Schedule Type** to **Scheduled** and set a frequency as per your needs. Hit the **Save** button:

The screenshot shows the 'Schedule' configuration dialog. It has a 'Schedule Type' section with 'Manual (Paused)' and 'Scheduled' radio buttons, where 'Scheduled' is selected and highlighted with a red box. Below it is a 'Schedule' section with dropdowns for 'Every' (Day), 'at' (22), ':' (00), and '(UTC+00:00) ...'. A checkbox for 'Show cron syntax' is checked. At the bottom are 'Cancel' and 'Save' buttons.

Figure 7.59 – Setting the schedule configuration

13. Hit **Run now** to trigger the job:

The screenshot shows the 'Job details' section of a Databricks job configuration. At the top right, there is a blue 'Run now' button with a dropdown arrow. Below it, the 'Job details' section includes fields for 'Job ID' (89), 'Creator' (arr.nagaraj@gmail.com), and 'Run as' (arr.nagaraj@gmail.com). The 'Schedule' section indicates a run at 10:00 PM (UTC+00:00 — UTC) with 'Edit schedule' and 'Pause' buttons.

Figure 7.60 – Run now

14. Click on **Runs** on the left tab to view the job run result:

[Jobs > SampleJob](#)

SampleJob



Figure 7.61 – Job runs

15. The **Active runs** section will provide the currently active jobs. After a few minutes, the job will move to the **Running** state from the initial **Pending** state:

Start time	Run ID	Launched	Duration	Spark	Status
Feb 19 2022, 21:54 PM +08	110	Manually	2m 54s	Spark UI / Logs / Metrics	Running

Figure 7.62 – Job runs

16. Upon completion, the job result will be listed as completed in the **Completed runs (past 60 days)** section. Click **Start time** to view the result of the completed notebook:

Completed runs (past 60 days)					Refresh
Latest successful run (refreshes automatically)					
Start time	Run ID	Launched	Duration	Spark	Status
Feb 19 2022, 21:54 PM +08	110	Manually	3m 22s	Spark UI / Logs / Metrics	Succeeded

Figure 7.63 – A completed run

17. The job execution result is shown in the following screenshot:

Jobs > SampleJob > Run 110

SampleJob run

- Output

```

val diamonds = spark.read.format("csv")
    .option("header", "true")
    .option("inferSchema", "true")
    .load("/databricks-datasets/Rdatasets/data-001/csv/ggplot2/diamonds.csv")

▶️ diamonds: org.apache.spark.sql.DataFrame = [_c0: integer, carat: double ... 9 more fields]
diamonds: org.apache.spark.sql.DataFrame = [_c0: int, carat: double ... 9 more fields]
Command took 21.47 seconds

```

```

diamonds.createOrReplaceTempView("diamonds_view")
Command took 0.30 seconds

```

```

%sql
Select cut, color, avg(price) as avg_price, max(price) as max_price
From diamonds_view
Group by cut,color
order by avg_price desc, cut, color

```

	cut	color	avg_price	max_price
1	Premium	J	6294.591584158416	18710
2	Premium	I	5946.180672268908	18823
3	Very Good	I	5255.879568106312	18500
4	Premium	H	5216.706779661017	18795
5	Fair	H	5135.683168316832	18565
6	Very Good	J	5103.513274336283	18430
7	Good	I	5078.532567049809	18707

Showing all 35 rows.

Command took 2.88 seconds

Figure 7.64 – The notebook output

18. If we need to execute another notebook at the same schedule, we could create an additional task within the same job. Go back to the job menu and click on **SampleJob**. Switch to the **Tasks** section. Click on the + button to add a new task:

Jobs > SampleJob

SampleJob

The screenshot shows the 'Tasks' tab selected in the 'SampleJob' interface. A red box highlights the '+ button' located at the bottom center of the task configuration area. The task configuration form includes fields for Type (Notebook), Cluster (New Job Cluster), Parameters, and Advanced options.

Figure 7.65 – Adding a new task

19. Provide a task name. You may add any notebook to the task. Notice that the cluster name is **SampleJob_cluster**, which implies that it will use the cluster created for the previous task. The **Depends on** option controls whether the job needs to run after the previous **SampleJob** task is completed:

The screenshot shows a form for creating a new task. The 'Task name' field contains 'Task2' and is highlighted with a red box. The 'Type' dropdown is set to 'Notebook' and has a value of '/Job/SampleJob'. The 'Cluster' dropdown is set to 'SampleJob_cluster' and is also highlighted with a red box. The 'Parameters' section includes a 'UI | JSON' link and an 'Add' button. The 'Depends on' section lists 'SampleJob' and has a remove button ('x'). At the bottom right are 'Cancel' and 'Create task' buttons, with the 'Create task' button highlighted with a red box.

Figure 7.66 – Adding a new task

20. Upon adding it, the two tasks will appear, as shown here:

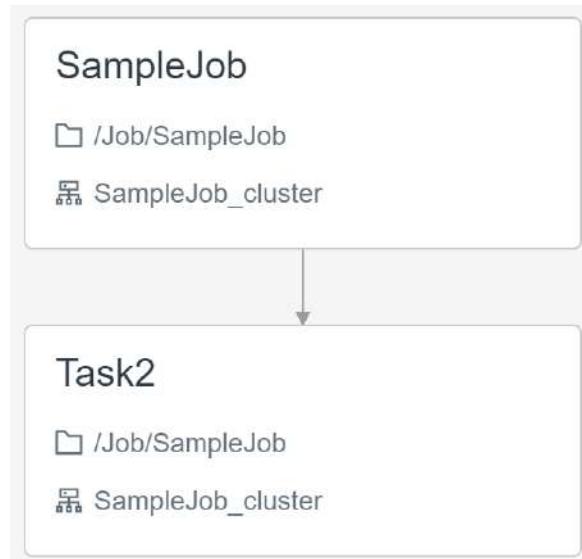


Figure 7.67 – Multiple tasks

How it works...

The imported notebook was set to run at a specific schedule using the Databricks job scheduling capabilities. While scheduling the jobs, **New Cluster** was selected, instead of picking any cluster available in the workspace. Picking **New Cluster** implies a cluster will be created each time the job runs and will be destroyed once the job completes. This also means the jobs need to wait for an additional 2 minutes for the cluster to be created for each run.

Adding multiple notebooks to the same job via additional tasks allows us to reuse the job cluster created for the first notebook execution, and the second task needn't wait for another cluster to be created. Usage of multiple tasks and the dependency option allows us to orchestrate complex data processing flows using Databricks notebooks.

Working with Delta Lake tables

Delta Lake databases are **Atomicity, Consistency, Isolation, and Durability** (ACID) property-compliant databases available in Databricks. Delta Lake tables are tables in Delta Lake databases that use Parquet files to store data and are highly optimized for performing analytic operations. Delta Lake tables can be used in a data processing notebook for storing preprocessed or processed data. The data stored in Delta Lake tables can be easily consumed in visualization tools such as Power BI.

In this recipe, we will create a Delta Lake database and Delta Lake table, load data from a CSV file, and perform additional operations such as UPDATE, DELETE, and MERGE on the table.

Getting ready

Create a Databricks workspace and a cluster, as explained in the *Configuring the Azure Databricks environment* recipe of this chapter.

Download the `covid-data.csv` file from this link: <https://github.com/PacktPublishing/Azure-Data-Engineering-Cookbook-2nd-edition/blob/main/chapter07/covid-data.csv>.

Upload the `covid-data.csv` file to the workspace, as explained in *step 1 to step 6* of the *How to do it...* section of the *Processing data using notebooks* recipe.

How to do it...

In this recipe, let's create a Delta Lake database and tables, and process data using the following steps:

1. From the Databricks menu, click **Create** and create a new notebook:

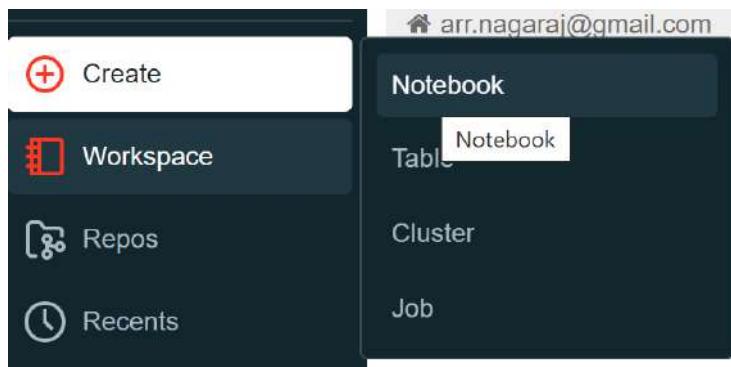


Figure 7.68 – A new notebook

2. Create a notebook called **Covid-DeltaTables** with **Default Language** set to **SQL**:

A screenshot of the 'Create Notebook' dialog box. It has a title bar 'Create Notebook' with a close button 'x'. Below the title, there is a 'Name' field containing 'Covid-DeltaTables'. Underneath it is a 'Default Language' dropdown menu set to 'SQL'. Below that is a 'Cluster' dropdown menu set to 'dbcluster01'. At the bottom right are two buttons: 'Cancel' and a blue 'Create' button.

Figure 7.69 – Delta notebook creation

3. Add a cell to the notebook and execute the following command to create a Delta database called **covid**:

```
CREATE DATABASE covid
```

The output is displayed in the following screenshot:

The screenshot shows the Databricks SQL interface. At the top, it says "Covid-DeltaTables" and "SQL". Below that is a toolbar with icons for cluster selection ("dbcluster01"), file operations ("File", "Edit", "View: Standard"), and a "Run All" button. The main area is titled "Cmd 1". It contains a single command: "CREATE DATABASE covid". Below the command is the output "OK". At the bottom, a status message says "Command took 0.75 seconds -- by arr.nagaraj@gmail.com at 19/02/2022, 23:52:07 on dbcluster01". There is also a "Cmd 2" section below.

Figure 7.70 – Delta database creation

4. Execute the following command to read the `covid-data.csv` file to a temporary view. Please note that the path depends on the location where `covid-data.csv` was uploaded and ensure to provide the correct path for your environment:

```
CREATE TEMPORARY VIEW covid_data
USING CSV
OPTIONS (path "/FileStore/shared_uploads/arr.nagaraj@gmail.com/covid_data.csv", header "true", mode "FAILFAST")
```

The output is displayed in the following screenshot:

The screenshot shows the Databricks SQL interface. At the top, it says "Cmd 2". Below that is a toolbar with icons for cluster selection ("dbcluster01"), file operations ("File", "Edit", "View: Standard"), and a "Run All" button. The main area is titled "Cmd 1". It contains three commands: "CREATE TEMPORARY VIEW covid_data", "USING CSV", and "OPTIONS (path "/FileStore/shared_uploads/arr.nagaraj@gmail.com/covid_data.csv", header "true", mode "FAILFAST")". Below the commands is the output "(1) Spark Jobs". At the bottom, a status message says "Command took 5.97 seconds -- by arr.nagaraj@gmail.com at 19/02/2022, 23:56:32 on dbcluster01".

Figure 7.71 – Temporary view creation

5. Execute the following command to create a Delta table using the temporary view. `USING DELTA` in the `CREATE TABLE` command indicates that it's a Delta table being created. The location specifies where the table is stored. If you have mounted a data lake container (as we did in the *Mounting an Azure Data Lake container in Databricks* recipe), you can use the data lake mount point to store the Delta Lake table in your Azure Data Lake account. In this example, we use the default storage provided by Databricks, which comes with each Databricks workspace.

The table name is provided in <database_name>. <table_name> to ensure it belongs to the Delta Lake database created. To insert the data from the view into the new table, the table is created using the CREATE TABLE command, followed by the AS command, followed by the SELECT statement against the view:

```
CREATE OR REPLACE TABLE covid.covid_data_delta
USING DELTA
LOCATION '/FileStore/shared_uploads/arr.nagaraj@gmail.com/covid_data_delta'
AS
SELECT iso_code,location,continent,date,new_deaths_per_million,people_fully_vaccinated,population FROM covid_data
```

The output is displayed in the following screenshot:

The screenshot shows a Databricks command line interface. The command entered is:

```
1 CREATE OR REPLACE TABLE covid.covid_data_delta
2 USING DELTA
3 LOCATION '/FileStore/shared_uploads/arr.nagaraj@gmail.com/covid_data_delta'
4 AS
5 SELECT iso_code,location,continent,date,new_deaths_per_million,people_fully_vaccinated,population FROM covid_data
```

Below the command, the output shows:

- ▶ (4) Spark Jobs

Query returned no results

Command took 13.18 seconds -- by arr.nagaraj@gmail.com at 28/02/2022, 00:04:30 on dbcluster01

Figure 7.72 – Creating the Delta table

6. Go to the Databricks menu, click **Data**, and then click on the **covid** database. You will notice that a **covid_data_delta** table has been created:

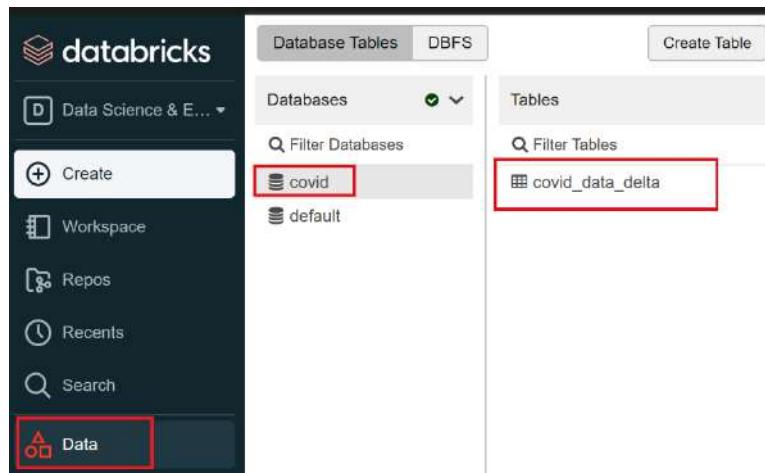


Figure 7.73 – The created Delta table

7. Go back to the notebook we were working on. Add a new cell and delete some rows using the following command. The `DELETE` command will delete around 57,000 rows. We can delete, select, and insert data as well as we would in any other commercial database:

```
DELETE FROM covid.covid_data_delta where population is
null or people_fully_vaccinated is null or new_deaths_
per_million is null or location is null
```

The output is displayed in the following screenshot:

```
Cmd 4
1 DELETE FROM covid.covid_data_delta where population is null or people_fully_vaccinated is null or new_deaths_per_million is null or location is null
▶ (6) Spark Jobs
Table Data Profile



|   | num_affected_rows |
|---|-------------------|
| 1 | 57075             |


Showing all 1 rows.
Command took 6.47 seconds -- by navenkat@microsoft.com at 28/08/2022, 11:56:56 on dbcluster01
```

Figure 7.74 – Deleting a few rows in a Delta table

8. Add a new cell and execute the following command to delete all the rows from the table. Let's run a `select count(*)` query against the table, which will return **0** if all the rows have been deleted:

```
delete from covid.covid_data_delta;
Select count(*) from covid.covid_data_delta;
```

The output is displayed in the following screenshot:

```
Cmd 5
1 delete from covid.covid_data_delta;
2 Select count(*) from covid.covid_data_delta;
▶ (3) Spark Jobs
Table Data Profile



|   | count(1) |
|---|----------|
| 1 | 0        |


Showing all 1 rows.
Command took 0.43 seconds -- by arr.nagaraj@gmail.com at 28/02/2022, 00:33:05 on dbcluster01
```

Figure 7.75 – Deleting all rows

9. Delta tables have the ability to time travel, which allows us to read older versions of the table. Using the `as of version <version number>` keyword, we can read the older version of the table. Version 0 gives the most recent version behind the current version of the table. Add a new cell and execute the following command to read the data before deletion:

```
select * from covid.covid_data_delta version as of 0;
```

The output is displayed in the following screenshot:

Cmd 6

```
1 select * from covid.covid_data_delta version as of 0;
```

(1) Spark Jobs

Table Data Profile

	iso_code	location	continent	date	new_deaths_per_million	people_fully_vaccinated	population
1	AFG	Afghanistan	Asia	1/1/2021	0.301	null	39835428
2	AFG	Afghanistan	Asia	2/1/2021	0.251	null	39835428
3	AFG	Afghanistan	Asia	3/1/2021	0.251	null	39835428
4	AFG	Afghanistan	Asia	4/1/2021	0.226	null	39835428
5	AFG	Afghanistan	Asia	5/1/2021	0.176	null	39835428
6	AFG	Afghanistan	Asia	6/1/2021	0.176	null	39835428
7	AFG	Afghanistan	Asia	7/1/2021	0.226	null	39835428

Truncated results, showing first 1000 rows.

Figure 7.76 – Check out an older version of the Delta table

10. RESTORE TABLE can restore the table to the older version. Add a cell and execute the following command to recover all the rows before deletion:

```
RESTORE TABLE covid_data_delta TO VERSION AS OF 0;
```

The output is displayed in the following screenshot:

Cmd 7

```
1 RESTORE TABLE covid_data_delta TO VERSION AS OF 0;
```

(19) Spark Jobs

Table Data Profile

table_size_after_restore	num_of_files_after_restore	num_removed_files	num_restored_files	removed_files_size	restored_files_size
1 8962891	4	0	0	0	0

Showing all 1 rows.

Command took 5.67 seconds -- by arr.nagaraj@gmail.com at 28/02/2022, 08:43:17 on dbcluster01

Figure 7.77 – Restoring the table to an older version

11. Let's perform an UPDATE statement, followed by a DELETE statement. Add two cells. Paste the following commands into these cells, as shown in the following screenshot. Execute them in sequence:

```
UPDATE covid_data_delta SET population = population * 1.2  
WHERE continent = 'Asia';  
DELETE FROM covid_data_delta WHERE continent = 'Europe';
```

The output is displayed in the following screenshot:

Cmd 8

```
1 update covid_data_delta set population = population * 1.2 where continent = 'Asia';
```

▶ (12) Spark Jobs

Table Data Profile

	num_affected_rows
1	20036

Showing all 1 rows.

Command took 6.25 seconds -- by arr.nagaraj@gmail.com at 20/02/2022, 00:49:30 on dbcluster01

Cmd 9

```
1 Delete from covid_data_delta where continent = 'Europe';
```

▶ (8) Spark Jobs

Table Data Profile

	num_affected_rows
1	20654

Showing all 1 rows.

Command took 4.07 seconds -- by arr.nagaraj@gmail.com at 20/02/2022, 00:49:47 on dbcluster01

Figure 7.78 – Update and delete

12. If we want to revert these two operations (UPDATE and DELETE), we can use the MERGE statement and the Delta table's row versioning capabilities to achieve this. Using a single MERGE statement, we can perform the following:

- Compare the older version of the table with the current version of the table
- If the rows match, update all the columns with values from the older version
- If the row doesn't match, insert the row from the older version into the current table

This can be achieved using the following code:

```
MERGE INTO covid_data_delta source
  USING covid_data_delta TIMESTAMP AS OF "2022-02-19
16:45:00" target
    ON source.location = target.location and source.date =
target.date
    WHEN MATCHED THEN UPDATE SET *
    WHEN NOT MATCHED
    THEN INSERT *
```

The MERGE command takes the `covid_data_delta` table as the source table to be updated or inserted. Instead of specifying version numbers, we can also use timestamps to obtain older versions of the table. `covid_data_delta TIMESTAMP AS OF "2022-02-19 16:45:00"` takes the version of the table as of February 19, 2022, 16:45:00 – UTC time. `WHEN MATCHED THEN UPDATE SET *` updates all the columns in the table when the condition specified in the ON clause matches. When the condition doesn't match, the rows are inserted from the older version to the current version of the table. The output, as expected, shows that the rows that were deleted in *step 11* were successfully reinserted:

The screenshot shows a terminal window titled 'Cmd 10' with the following content:

```
1 | MERGE INTO covid_data_delta source
2 | USING covid_data_delta TIMESTAMP AS OF "2022-02-19 16:45:00" target
3 | ON source.location = target.location and source.date = target.date
4 | WHEN MATCHED THEN UPDATE SET *
5 | WHEN NOT MATCHED
6 | THEN INSERT *
```

Below the command, it says '(13) Spark Jobs'. Under 'Table Data Profile', there is a table:

	num_affected_rows	num_updated_rows	num_deleted_rows	num_inserted_rows
1	94342	73688	0	20654

Showing all 1 rows.

Command took 12.56 seconds -- by arr.nagaraj@gmail.com at 28/02/2022, 00:56:26 on dbcluster01

Figure 7.79 – The merge statement

How it works...

Delta tables offer advanced capabilities for processing data, such as support for UPDATE, DELETE, and MERGE statements. MERGE statements and the versioning capabilities of Delta tables are very powerful in ETL scenarios, where we need to perform UPSERT (update if it matches, insert if it doesn't) operations against various tables.

These capabilities for supporting data modifications and row versioning are made possible because Delta tables maintain the changes to the table via a transaction log file stored in JSON format. The transaction files are located in the same location where the table was created but in a subfolder called `_delta_log`. By default, the log files are retained for 30 days and can be controlled using the `delta.logRetentionDuration` table property. The ability to read older versions is also controlled by the `delta.logRetentionDuration` property.

Connecting a Databricks Delta Lake table to Power BI

Delta Lake databases are commonly used to store processed data in Delta Lake tables, which is then ready to be consumed by reporting-layer applications such as Power BI. Delta Lake tables are best suited for handling analytic workloads from Power BI, as Delta Lake tables use Parquet files as storage, which offer optimal performance for analytic workloads.

In this recipe, we will use Power BI Desktop, connect to a Delta Lake table, and build a simple report in Power BI.

Getting ready

Create a Databricks workspace and a cluster as explained in the *Configuring the Azure Databricks environment* recipe.

Download the latest version of Power BI Desktop from <https://powerbi.microsoft.com/en-us/downloads/> and install Power BI Desktop on your machine.

How to do it...

Perform the following steps to connect a Delta Lake table to a Power BI report and create visualizations:

1. Log in to the Databricks portal and click on the **Compute** button in the menu:

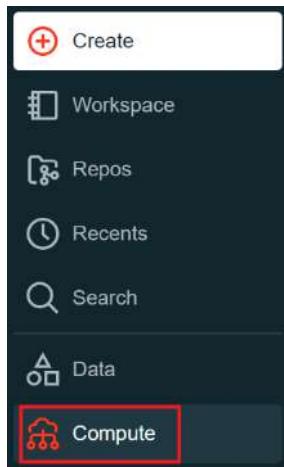


Figure 7.80 – Compute

2. Click on the cluster and start it if the cluster is terminated. Click on **Advanced options**:

A screenshot of the Databricks Cluster configuration page for cluster dbcluster01. The top navigation bar shows 'Clusters / dbcluster01'. Below the cluster name are buttons for Edit, Start (which is highlighted with a red box), Clone, and Delete. A horizontal line separates this from the configuration details. The 'Configuration' tab is selected. The 'Databricks Runtime Version' is set to '9.1 LTS (Includes Apache Spark 3.1.2, Scala 2.12)'. Under 'Autopilot options', there are two checked checkboxes: 'Enable autoscaling' and 'Terminate after 10 minutes of inactivity'. In the 'Worker type' section, 'dbclusterpool' is selected for both Driver type and Worker pool, with 'Standard_DS3_v2' chosen for both. The 'Min workers' and 'Max workers' fields show values of 1 and 2 respectively. At the bottom left, a button labeled 'Advanced options' is highlighted with a red box.

Figure 7.81 – Advanced options for the cluster

3. Click on the **JDBC/ODBC** section. Copy the values from the **Server Hostname** and **HTTP Path** sections. We will need them when connecting to the Delta Lake table using Power BI Desktop:

▼ Advanced options

Azure Data Lake Storage credential passthrough ? Available on Azure Databricks premium [Learn more](#)

Enable credential passthrough for user-level data access

Spark Tags Logging Init Scripts **JDBC/ODBC** Permissions

Server Hostname
adb-7675839323985314.14.azuredatabricks.net

Port
443

Protocol
HTTPS

HTTP Path
sql/protocolv1/o/7675839323985314/0212-043423-8h4nmkpl

JDBC URL ?
jdbc:spark://adb-7675839323985314.14.azuredatabricks.net:443/default;transportMode=http;ssl=1;httpPath=sql/protocolv1/o/7675839323985314/0212-043423-8h4nmkpl;AuthMech=3;UID=token;PWD=<personal-access-token>

Figure 7.82 – The JDBC/ODBC connection string

4. Click on the **Workspace** button in the Databricks menu. Click on any folder (by default under your username) and click on the **Import** button:

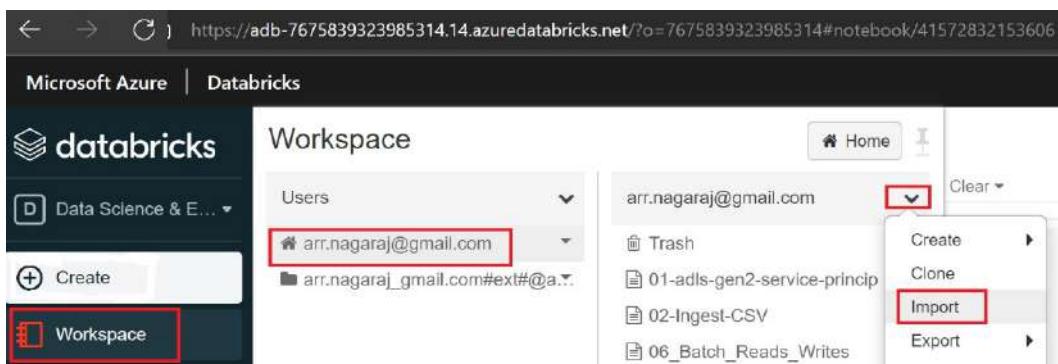


Figure 7.83 – Importing the notebook

5. For the **Import from** option, pick **URL**. Provide https://github.com/PacktPublishing/Azure-Data-Engineering-Cookbook-2nd-edition/blob/main/chapter07/Delta_PowerBIdbc as the input and click the **Import** button:

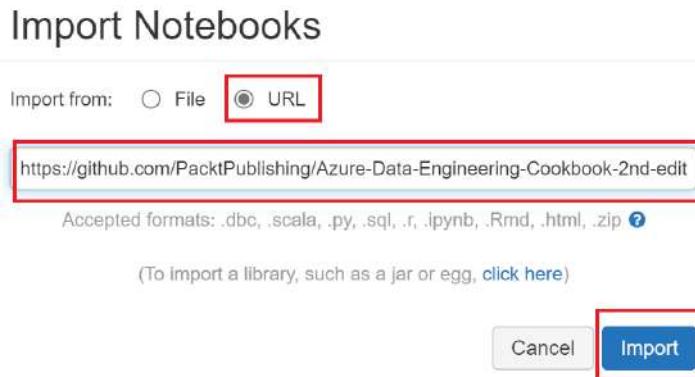


Figure 7.84 – Import notebook

6. This notebook will read a CSV file, create a database called **PowerBI**, and create a Delta table called **Diamond_Insights_PowerBI**. Click on the **Run All** button in the top-right corner of the notebook:

The screenshot shows a Databricks notebook titled 'Delta_PowerBI'. The notebook has four command cells labeled Cnd 1 through Cnd 4. In Cnd 1, Scala code reads a CSV file into a DataFrame named 'diamonds'. In Cnd 2, the DataFrame is registered as a temporary view named 'diamonds_view'. In Cnd 3, a database named 'PowerBI' is created if it does not exist. In Cnd 4, a Delta table named 'Diamond_Insights_PowerBI' is created using the 'diamonds_view'. The 'Run All' button is located at the top right of the notebook interface, and it is highlighted with a red box.

```
%scala
val diamonds = spark.read.format("csv")
  .option("header", "true")
  .option("inferSchema", "true")
  .load("/databricks-datasets/Rdatasets/data-001/csv/ggplot2/diamonds.csv")

diamonds: org.apache.spark.sql.DataFrame = [c0: integer, carat: double ... 9 more fields]
diamonds: org.apache.spark.sql.DataFrame = [_c0: int, carat: double ... 9 more fields]

Command took 28.85 seconds -- by a user at 28/02/2022, 16:10:49 on unknown cluster
```

```
%sql
diamonds.createOrReplaceTempView("diamonds_view")
```

```
CREATE DATABASE IF NOT EXISTS PowerBI
```

```
%sql
CREATE TABLE PowerBI.Diamond_Insights_PowerBI
USING DELTA
AS
Select *
From diamonds_view
```

Figure 7.85 – Run all cells

7. Open the installed **Power BI Desktop**:

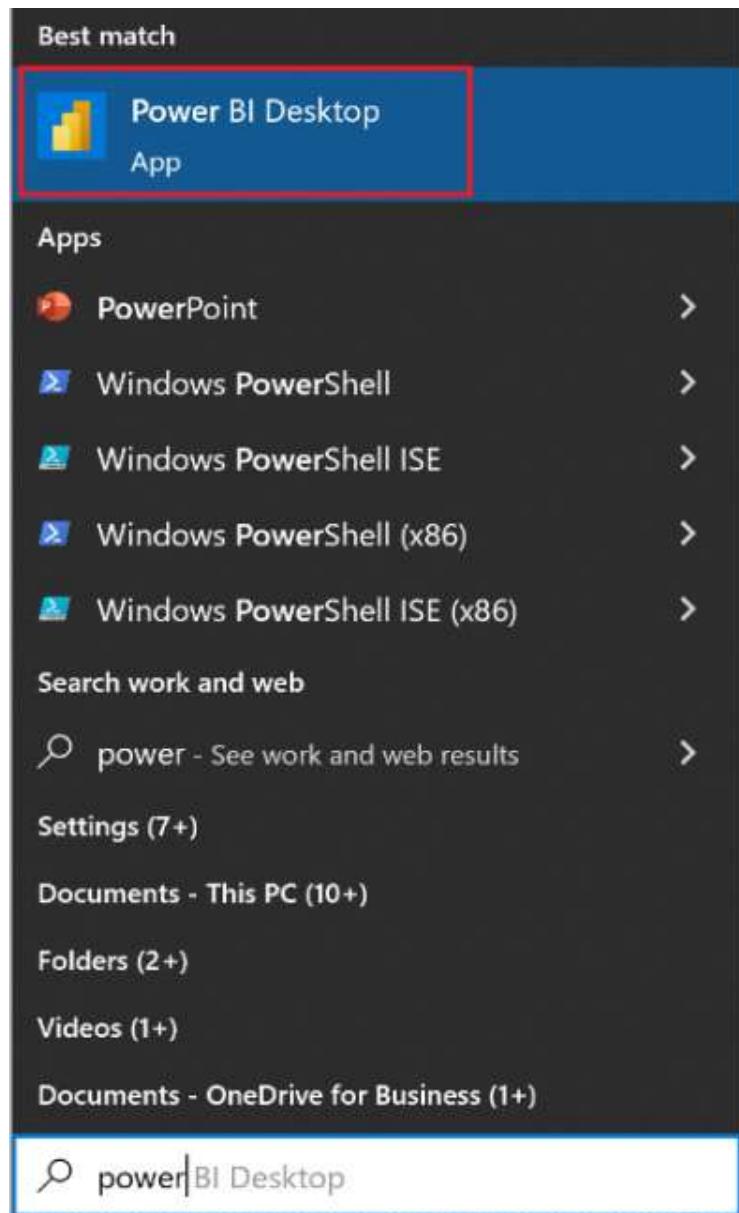


Figure 7.86 – Start Power BI Desktop

8. Click **Get data** and then **More...**:

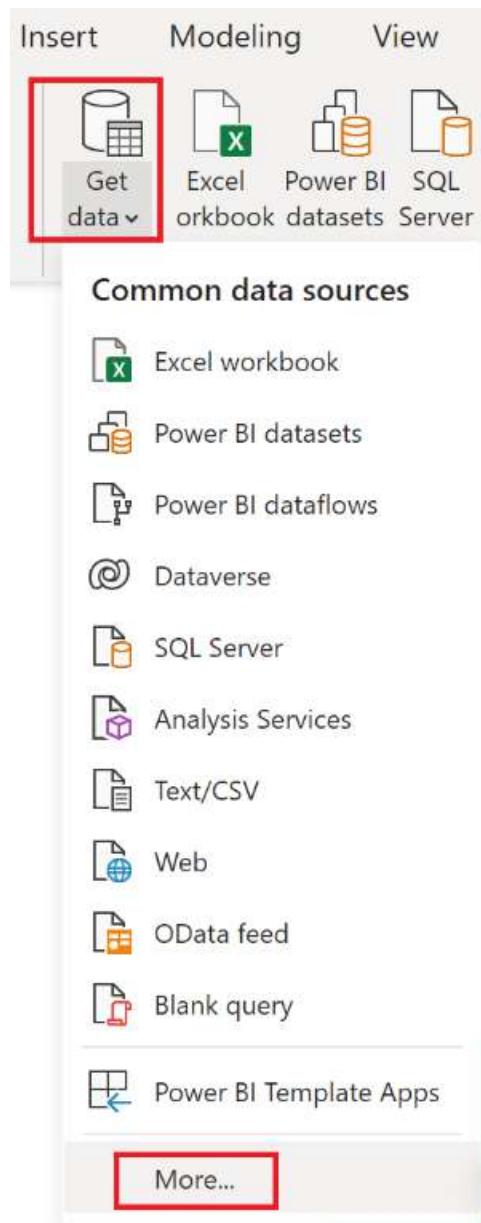


Figure 7.87 – Starting Power BI Desktop

9. Select **Azure Databricks** and click on the **Connect** button:

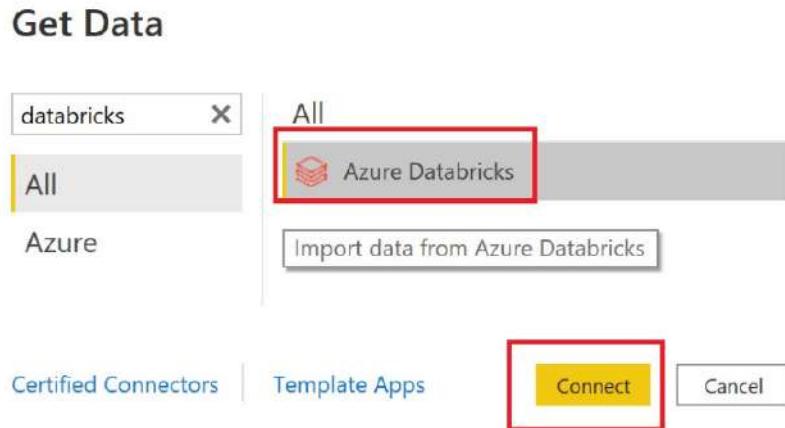


Figure 7.88 – Connecting to Databricks

10. Provide the **Server Hostname** value and the **HTTP Path** value noted in *step 3*, and click on the **Ok** button:

The screenshot shows the 'Azure Databricks' connection configuration dialog. It has two main input fields: 'Server Hostname' containing 'adb-7675839323985314.14.azuredatabricks.net' and 'HTTP Path' containing 'sql/protocolv1/o/7675839323985314/0212-043423-8h4nmkpl'. Both of these fields are highlighted with red rectangular boxes. Below these fields are optional sections: 'Advanced Options (optional)', 'Default catalog (optional)', and 'Database (optional)'. The 'Default catalog' field contains the placeholder text 'Example: abc'.

Figure 7.89 – Connecting to the Delta database

-
11. Sign in to Azure using the Azure account that you used to create the Databricks workspace:

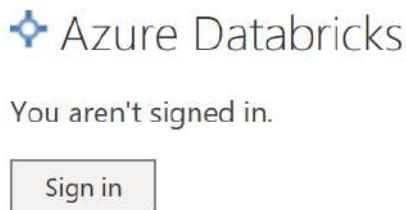


Figure 7.90 – Connecting to Azure

12. Once you're signed in, press the **Connect** button:

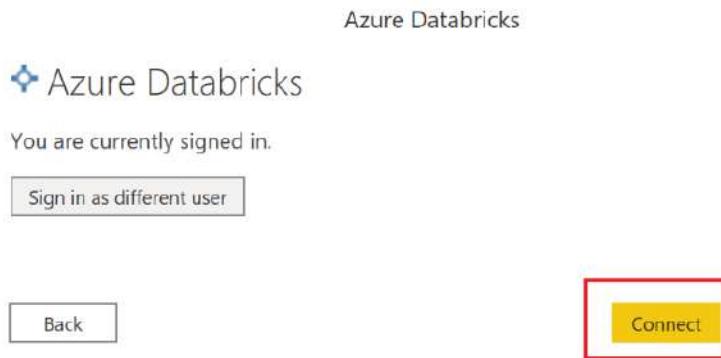


Figure 7.91 – Connecting to the Delta table

13. Expand **hive_metastore** and then the **powerbi** database. Select the **diamond_insights_powerbi** table and click on the **Load** button:

The screenshot shows the Databricks Navigator interface. On the left, the file tree displays a folder structure under 'hive_metastore [3]'. A red box highlights the 'powerbi [1]' folder, which contains a sub-item 'diamond_insights_powerbi'. This item also has a red box around it. On the right, a preview of the 'diamond_insights_powerbi' table is shown in a tabular format with columns: _c0, carat, cut, color, clarity, depth, and table. The table contains 23 rows of diamond data. At the bottom right of the preview area, there are three buttons: 'Load' (highlighted with a red box), 'Transform Data', and 'Cancel'.

_c0	carat	cut	color	clarity	depth	table
1	0.23	Ideal	E	SI2	61.5	diamond_insights_powerbi
2	0.21	Premium	E	SI1	59.8	
3	0.23	Good	E	VS1	56.9	
4	0.29	Premium	I	VS2	62.4	
5	0.31	Good	J	SI2	63.3	
6	0.24	Very Good	J	VVS2	62.8	
7	0.24	Very Good	I	VVS1	62.3	
8	0.26	Very Good	H	SI1	61.9	
9	0.22	Fair	E	VS2	65.1	
10	0.23	Very Good	H	SI1	59.4	
11	0.3	Good	J	SI1	64	
12	0.23	Ideal	J	VS1	62.8	
13	0.22	Premium	F	SI1	60.4	
14	0.31	Ideal	J	SI2	62.2	
15	0.2	Premium	E	SI2	60.2	
16	0.32	Premium	E	I1	60.9	
17	0.3	Ideal	I	SI2	62	
18	0.3	Good	J	SI1	63.4	
19	0.3	Good	J	SI1	63.8	
20	0.3	Very Good	J	SI1	62.7	
21	0.3	Good	I	SI2	63.3	
22	0.23	Very Good	E	VS2	63.8	
23	0.23	Very Good	H	VVS1	61	

Figure 7.92 – Loading the Delta table

14. Select the `color` and `price` columns on the right-hand side. Set the **Visualizations** type to **Clustered column chart**. This will show the price of diamonds by color code. This will indicate that **color code G** has the highest price:

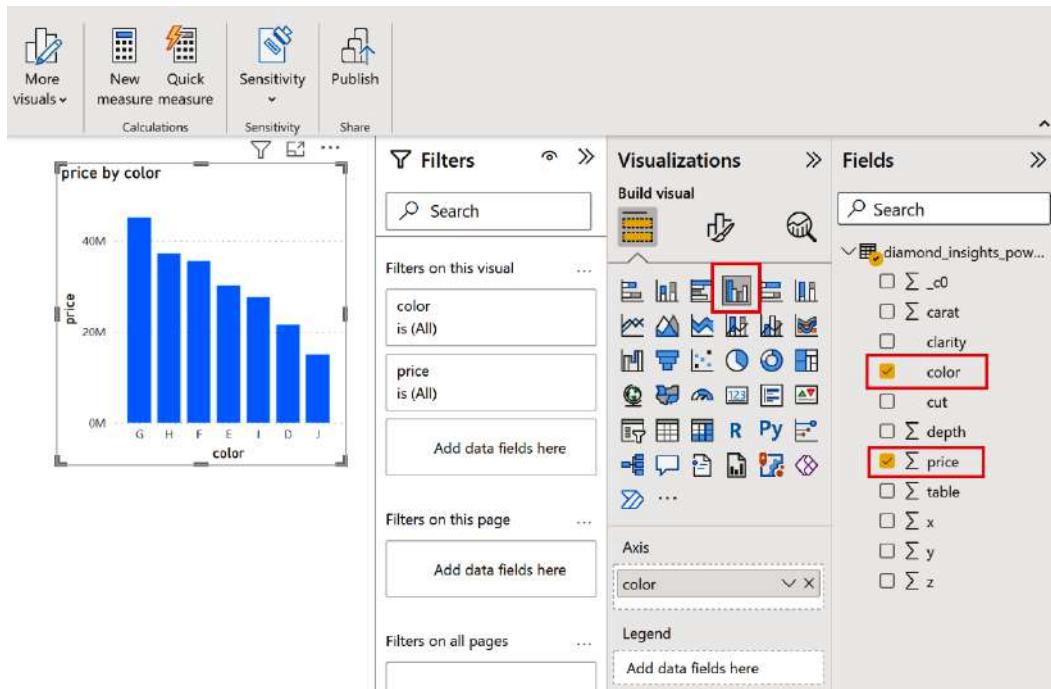


Figure 7.93 – Adding a clustered column chart

- Click on the white space anywhere outside of the clustered column chart visual. Click on the **cut** and **price** columns and set the **Visualizations** type to **Clustered column chart**. Now, you will have added another visual:

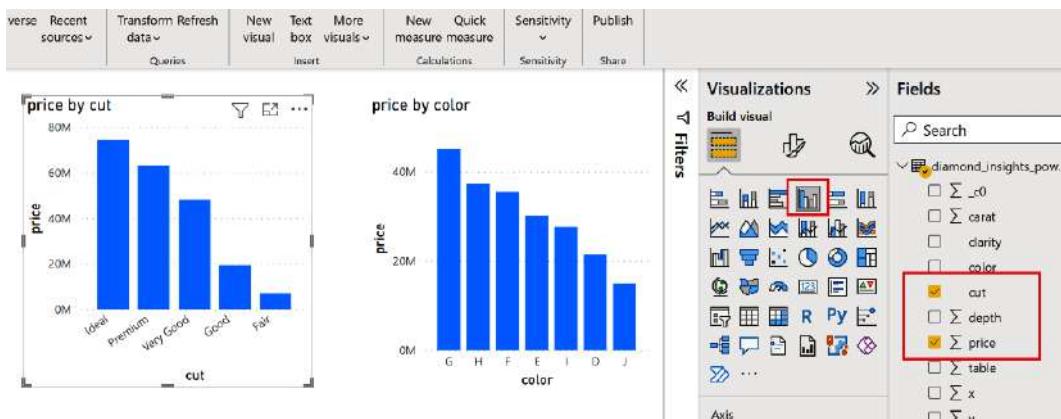


Figure 7.94 – Adding another visual

16. Click on the bar against the **Ideal** value on the **price by cut** visual. The **price by color** visual will be automatically filtered and will show that **color G** contributes close to 20 million in price when the cut is **Ideal**:

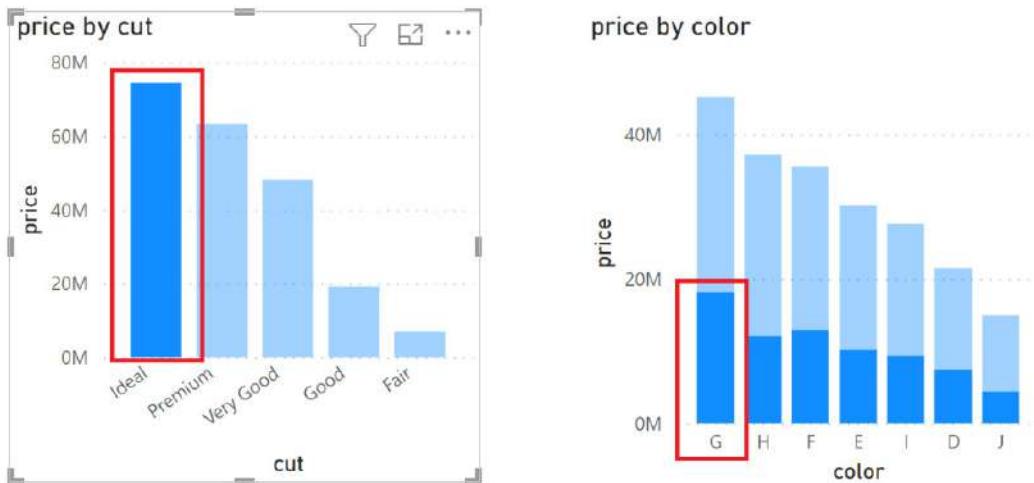


Figure 7.95 – Get insights using visuals

17. Open the **File** menu and hit the **Save** button to save the Power BI report.

How it works...

We used the Azure AD credentials to sign in to Azure Databricks and extracted the connection string from the Databricks cluster. When the connection request comes from Power BI, Databricks authenticates using Azure AD credentials. For the authentication to succeed, the Databricks cluster needs to be up and running. Once authenticated, Delta Lake tables are accessible, just as with any other database tables using Power BI. We added two simple visuals to explore the data visually. Clicking on one of the visuals automatically filters the other visual, which allows us to get insights out of the data easily.

8

Processing Data Using Azure Synapse Analytics

Azure Synapse Analytics workspaces generation 2, formally released in December 2020, is the industry-leading big data solution for processing and consolidating data of business value. Azure Synapse Analytics has three important components:

- SQL pools and Spark pools for performing data exploration and processing
- Integration pipelines for performing data ingestion and data transformations
- Power BI integration for data visualization

Having data ingestion, processing, and visualization capabilities in a single service with seamless integration with all other services makes Azure Synapse Analytics a very powerful tool in big data engineering projects. This chapter will introduce you to Synapse workspaces and cover the following recipes:

- Provisioning an Azure Synapse Analytics workspace
- Analyzing data using serverless SQL pool
- Provisioning and configuring Spark pools
- Processing data using Spark pools and a lake database
- Querying the data in a lake database from serverless SQL pool
- Scheduling notebooks to process data incrementally
- Visualizing data using Power BI by connecting to serverless SQL pool

By the end of the chapter, you will have learned how to provision a Synapse workspace and Spark pools, explore and analyze data using serverless SQL pool and Spark pools, create a lake database and query a lake database from serverless SQL pool and a Spark pool, and finally, visualize the lake database data in Power BI.

Technical requirements

For this chapter, you will need the following:

- A Microsoft Azure subscription
- PowerShell 7
- Power BI Desktop

Provisioning an Azure Synapse Analytics workspace

In this recipe, we'll learn how to provision a Synapse Analytics workspace. A Synapse Analytics workspace is the logical container that will hold the Spark pools, SQL pool, and integration pipelines that are required for the data engineering tasks.

Getting ready

To get started, log into <https://portal.azure.com> using your Azure credentials.

How to do it...

Follow these steps to create a Synapse Analytics workspace:

1. Go to the Azure portal home page at portal.azure.com and click on **Create a resource**. Search for Synapse and select **Azure Synapse Analytics**. Click the **Create** button:

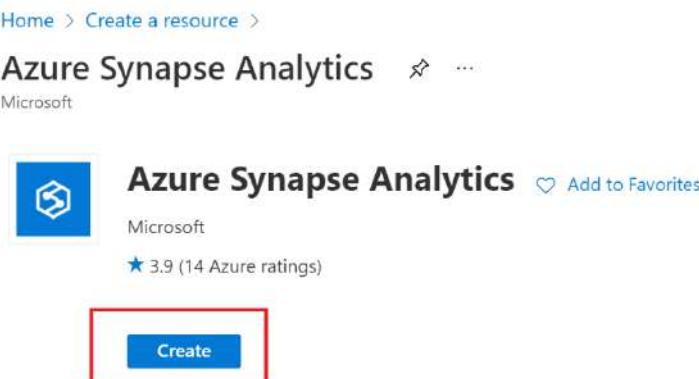


Figure 8.1 – Creating a Synapse Analytics workspace

2. Provide the following details:
 - I. Create a new resource group called **PacktADESynapse** by clicking the **Create new** link.

- II. Provide a unique workspace name. For this example, we are using packtadesynapse. You may pick any location.
- III. Provisioning a Synapse Analytics workspace requires an Azure Data Lake Storage Gen2 account. You may use an existing account or create a new account using the **Create new** link. Let's create a new Azure Data Lake Storage Gen2 account called packatadesynapse by clicking the **Create new** link.
- IV. A Synapse Analytics workspace requires a container on the new data lake account provisioned. Let's create a new container named synapse by clicking the **Create new** link. After the Synapse workspace is provisioned, the Synapse workspace service account is given permissions on the data lake account and the container by default. We will be able to connect to the data stored in the storage account from the Synapse workspace seamlessly. The data lake account will be used to store a lake database (a component of Synapse Spark pools) and other artifacts of the Synapse workspace. Click the **Next: Security >** button at the bottom:

Home > Create a resource > Azure Synapse Analytics >

Create Synapse workspace

Project details

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

Subscription *

Resource group *

Managed resource group

Workspace details

Name your workspace, select a location, and choose a primary Data Lake Storage Gen2 file system to serve as the default location for logs and job output.

Workspace name *

Region *

Select Data Lake Storage Gen2 * From subscription Manually via URL

Account name *

File system name *

Assign myself the Storage Blob Data Contributor role on the Data Lake Storage Gen2 account to interactively query it in the workspace.

Info We will automatically grant the workspace identity data access to the specified Data Lake Storage Gen2 account, using the [Storage Blob Data Contributor](#) role. To enable other users to use this storage account after you create your workspace, perform these tasks:

- Assign other users to the **Contributor** role on workspace
- Assign other users the appropriate [Synapse RBAC roles](#) using Synapse Studio

Review + create **< Previous** **Next: Security >**

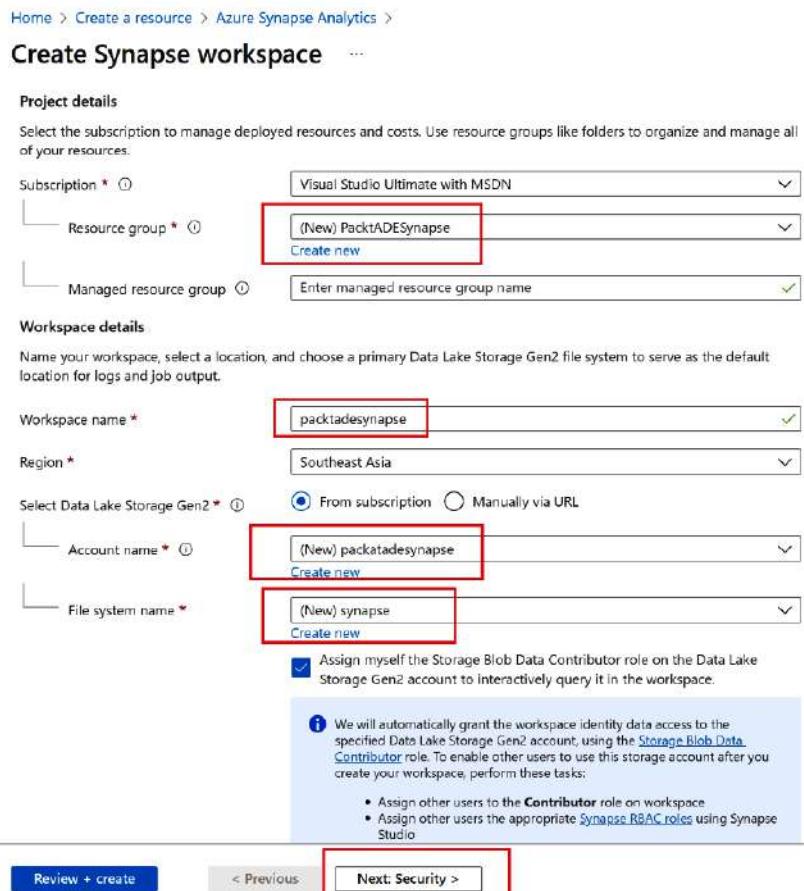


Figure 8.2 – Setting up Project details in a Synapse Analytics workspace

- V. On occasion, Azure subscriptions may not be able to create a Synapse workspace because of a **Resource provider not registered for the subscription** error. To resolve the error, go to the Azure portal and open your subscription. Click the **Resource providers** section, search for the Synapse resource, and register **Microsoft.Synapse** on the subscription. Please refer to <https://docs.microsoft.com/en-us/azure/azure-resource-manager/management/resource-providers-and-types#azure-portal> for detailed steps on how to register a resource on a subscription.
3. On the **Security** page, we will be filling in the **user ID** and **password** details for SQL pool. Provide the user ID and password as `sqladminuser` and `PacktAdeSynapse123`. Click **Review + Create** to create the Synapse Analytics workspace:

The screenshot shows the 'Create Synapse workspace' configuration page in the Azure portal, specifically the 'Security' tab. The page title is 'Create Synapse workspace' with a '...' button. Below it, there are tabs: 'Basics' (disabled), 'Security' (selected and underlined), 'Networking', 'Tags', and 'Review + create'. A note says 'Configure security options for your workspace.'

Authentication

Choose the authentication method for access to workspace resources such as SQL pools. The authentication method can be changed later on. [Learn more](#)

Authentication method: Use both local and Azure Active Directory (Azure AD) authentication Use only Azure Active Directory (Azure AD) authentication

SQL Server admin login *	<input type="text" value="sqladminuser"/>
SQL Password	<input type="password" value="*****"/> ✓
Confirm password	<input type="password" value="*****"/> ✓ Pass

System assigned managed identity permission

Select to grant the workspace network access to the Data Lake Storage Gen2 account using the workspace system identity. [Learn more](#)

Allow network access to Data Lake Storage Gen2 account. ⓘ The selected Data Lake Storage Gen2 account does not restrict network access using any network access rules, or you selected a storage account manually via URL under Basics tab. [Learn more](#)

Workspace encryption

⚠ Double encryption configuration cannot be changed after opting into using a customer-managed key at the time of workspace creation.

Choose to encrypt all data at rest in the workspace with a key managed by you (customer-managed key). This will provide double encryption with encryption at the infrastructure layer that uses platform-managed keys. [Learn more](#)

Enable Disable

Review + create < Previous Next: Networking >

Figure 8.3 – Creating SQL pool in a Synapse Analytics workspace

The preceding steps will create a Synapse Analytic workspace and a storage account. In the subsequent recipes of this chapter, we will use the workspace to explore and process data using SQL pool and Spark pool.

Analyzing data using serverless SQL pool

Serverless SQL pool allows us to explore data using T-SQL commands in a Synapse Analytics workspace. The key advantage of serverless SQL pool is that it is available by default once a Synapse Analytics workspace is provisioned with no cluster or additional resources to be created. In serverless SQL pool, you will be charged only for the data processed by the queries as it is designed as a pure pay-per-use model.

Getting ready

Create a Synapse Analytics workspace, as explained in the *Provisioning an Azure Synapse Analytics workspace* recipe of this chapter.

How to do it...

In this recipe, we will perform the following:

- Uploading sample data into a Synapse Analytics workspace data lake account and querying it using serverless SQL pool
- Creating a serverless SQL database and defining a view to read the data stored in the data lake account

The detailed steps to perform these tasks are as follows:

1. Log in to `portal.azure.com`, go to **All resources**, and search for `packtadesynapse`, the Synapse Analytics workspace we created in the *Provisioning an Azure Synapse Analytics workspace* recipe. Click the workspace. Click **Open Synapse Studio**:

The screenshot shows the Azure portal interface for a Synapse workspace named 'packtadesynapse'. The left sidebar contains navigation links for Overview, Activity log, Access control (IAM), Tags, Diagnose and solve problems, Settings (Azure Active Directory, Properties, Locks), Analytics pools (SQL pools, Apache Spark pools, Data Explorer pools (preview)), and Security (Encryption). The main content area displays the 'Essentials' section with resource details:

Resource group (move)	:	PacktADESynapse
Status	:	Succeeded
Location	:	
Subscription (move)	:	Visual Studio Ultimate wit
Subscription ID	:	
Managed virtual network	:	Yes
Managed Identity object ...	:	
Workspace web URL	:	https://web.azuresynapse.net/
Tags (edit)	:	Click here to add tags

The 'Getting started' section features a callout box with the heading 'Open Synapse Studio' and the subtext 'Start building your fully-integrated analytics solution and unlock new insights.' A blue 'Open' button is located at the bottom of the callout.

Figure 8.4 – Open Synapse Studio

2. Click on the blue cylinder (the data symbol) on the left, which will take you to the **Data** section. Click the **Linked** tab. Expand the data lake account of the Synapse workspace (packtadesynapse for this example) and click on the **synapse (Primary)** container. Click the **+ New folder** button and create a folder called CSV:

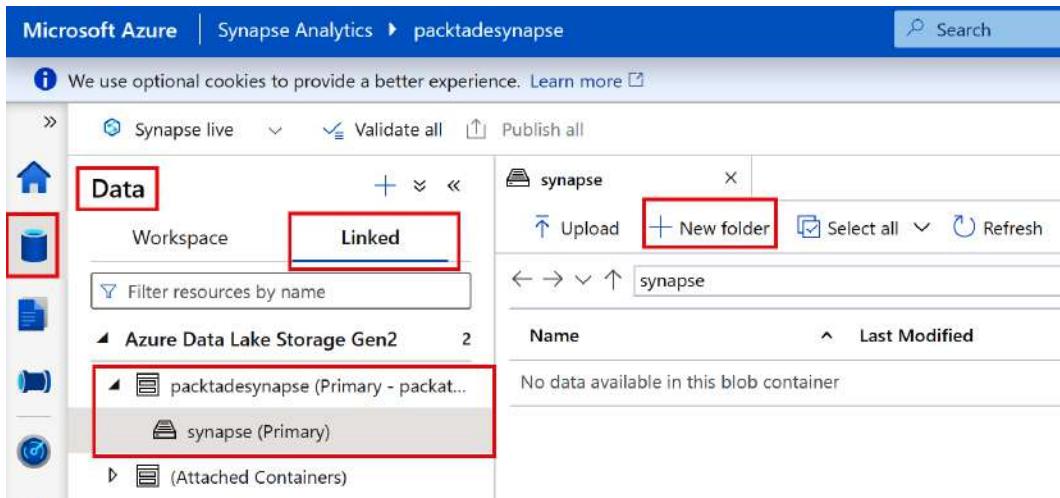


Figure 8.5 – Synapse studio folder creation

3. Download the `covid-data.csv` file from <https://github.com/PacktPublishing/Azure-Data-Engineering-Cookbook-2nd-edition/blob/main/chapter07/covid-data.csv> to your local machine.
4. Double click on the CSV folder in Synapse Studio. Click on the **Upload** button and upload the `covid-data.csv` file from your local machine into the data lake account of the Synapse workspace:

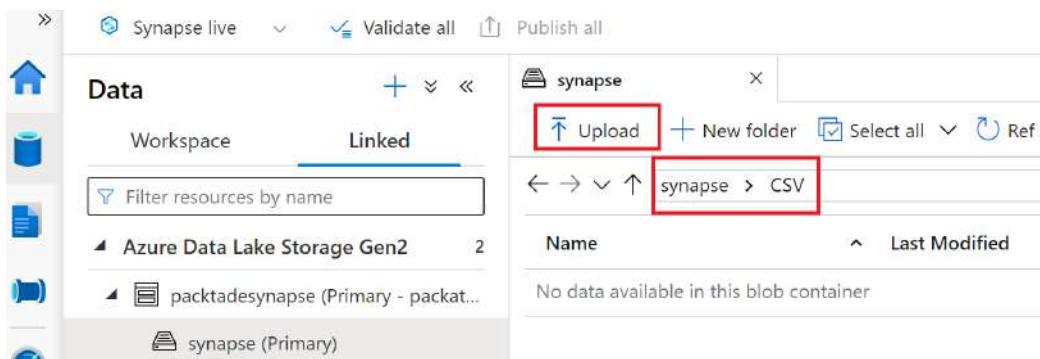


Figure 8.6 – Synapse Studio folder creation

5. After the file has uploaded, right-click the `covid-data.csv` file in the CSV folder, select **New SQL script**, and click the **Select TOP 100 rows** option:

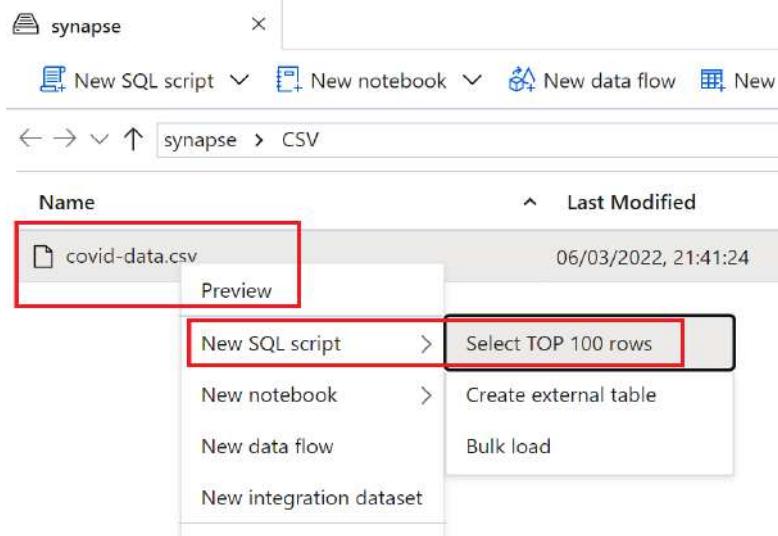


Figure 8.7 – Synapse Studio folder creation

6. A new query window will open with a script using the OPENROWSET command. Click the **Run** button to execute the query and preview the data:

```

1 -- This is auto-generated code
2 SELECT
3     TOP 100 *
4 FROM
5     OPENROWSET(
6         BULK 'https://packtadesynapse.dfs.core.windows.net/synapse/CSV/covid-data.csv',
7         FORMAT = 'CSV',
8         PARSE_VERSION = '2.0'
9     ) AS [result]
10

```

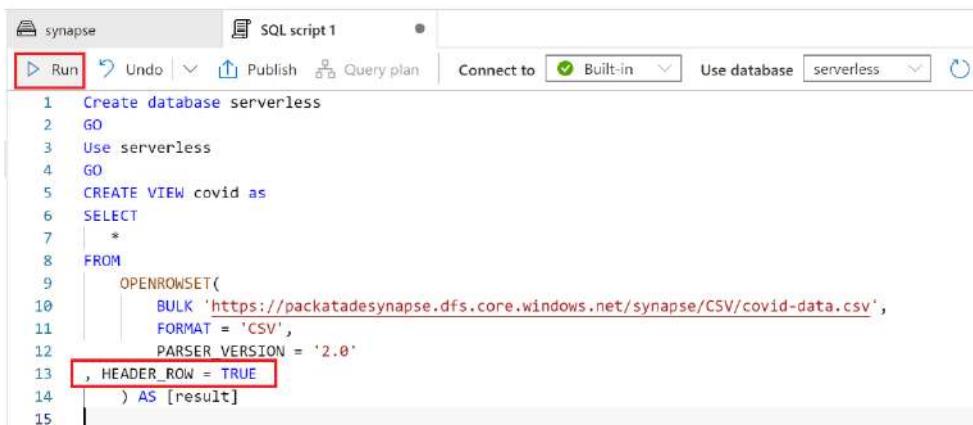
C1	C2	C3	C4	C5	C6	C7	C8
iso_code	continent	location	date	total_cases	new_cases	new_cases_smoothed	total_deaths
AFG	Asia	Afghanistan	1/1/2021	52513	183	131,143	2201
AFG	Asia	Afghanistan	2/1/2021	52586	73	117,429	2211
AFG	Asia	Afghanistan	3/1/2021	52709	123	123	2221
AFG	Asia	Afghanistan	4/1/2021	52909	200	128,857	2230
AFG	Asia	Afghanistan	5/1/2021	53011	102	123,429	2237

Figure 8.8 – Querying data using OPENROWSET

7. Let's perform the following. Let's create a **serverless database** and a **view** referencing the OPENROWSET command to read the covid-data.csv file. We notice that the actual column names (iso_code, continent, location, and date) are listed in the first row. These columns (iso_code, continent, location, and date) need to move up to become the table's column names and we need to remove the existing column names (C1, C2, C3, C4, and so on). We can fix that by adding the HEADER_ROW = TRUE option after the PARSER_VERSION option in the OPENROWSET command. Use the following command to create a database, create a view, and fix the header:

```
CREATE DATABASE serverless
GO
USE serverless
GO
CREATE VIEW covid AS
SELECT
*
FROM
OPENROWSET(
    BULK 'https://packatadesynapse.dfs.core.windows.net/synapse/CSV/covid-data.csv',
    FORMAT = 'CSV',
    PARSER_VERSION = '2.0'
    , HEADER_ROW = TRUE
) AS [result]
```

The result of the query execution is demonstrated here:



```
synapse SQL script1
Run Undo Publish Query plan Connect to Built-in Use database serverless
1 Create database serverless
2 GO
3 Use serverless
4 GO
5 CREATE VIEW covid AS
6 SELECT
7 *
8 FROM
9 OPENROWSET(
10     BULK 'https://packatadesynapse.dfs.core.windows.net/synapse/CSV/covid-data.csv',
11     FORMAT = 'CSV',
12     PARSER_VERSION = '2.0'
13     , HEADER_ROW = TRUE
14 ) AS [result]
15
```

Figure 8.9 – Create view in serverless SQL

- Click on the icon that looks like a notebook on the left-hand side of the screen. This will take you to the **Develop** section. Click the + button on top and click **SQL script**:

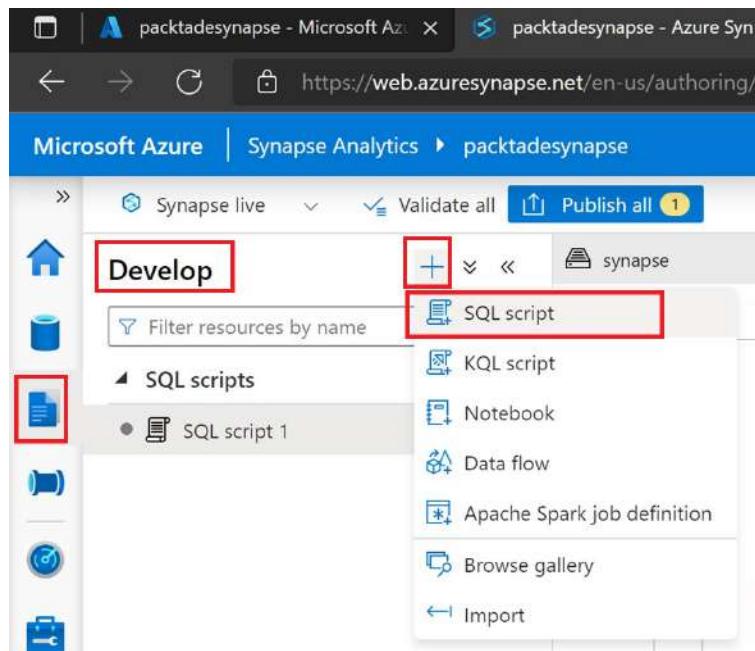


Figure 8.10 – A new SQL script

- Use the following script, referencing the serverless database and the view created, to find a list of countries that have the maximum number of deaths per million people on a given day:

```
use serverless
GO
Select iso_code,location , continent,
max(isnull(new_deaths_per_million,0)) as death_sum,
max(isnull(people_fully_vaccinated,0) / isnull(population,0)) * 100 as percentage_vaccinated From covid
where isnull(population,0) > 1000000
group by iso_code,location,continent
order by death_sum desc
```

The output of the preceding query is shown in the following screenshot:

The screenshot shows the Azure Synapse Studio interface. At the top, there are two tabs: 'synapse' and 'SQL script 1'. Below them are buttons for 'Run', 'Undo', 'Publish', 'Query plan', 'Connect to', 'Built-in', 'Use database', and 'serverless'. The 'serverless' option is selected. The main area contains a SQL script:

```
1 use serverless
2 GO
3 Select iso_code,location , continent,
4 max(isnull(new_deaths_per_million,0)) as death_sum,
5 max(isnull(people_fully_vaccinated,0) / isnull(population,0)) * 100 as percentage_vaccinated From covid
6 where isnull(population,0) > 1000000
7 group by iso_code,location,continent
8 order by death_sum desc
9
```

Below the script, there are tabs for 'Results' and 'Messages', with 'Results' being active. Under 'View', the 'Table' tab is selected. The results table displays the following data:

iso_code	location	continent	death_sum	percentage_va..
KAZ	Kazakhstan	Asia	120.611	0
LSO	Lesotho	Africa	106.527	0
BIH	Bosnia and Herzegovina	Europe	63.43	0
BWA	Botswana	Africa	58.818	0
NAM	Namibia	Africa	57.975	0
LBN	Lebanon	Asia	51.853	0

Figure 8.11 – Querying the serverless view

How it works...

After we uploaded the `covid-data.csv` file to the Azure Data Lake Storage account associated with the Synapse workspace, we were able to query the data at the click of a button, without providing any credentials or provisioning any other resources. Serverless SQL pool, which is available by default in a Synapse workspace, allows us to interact with the data with minimal effort.

We used the `OPENROWSET` function to read the data from a CSV file and we encapsulated it inside a view for easier access in subsequent scripts. The serverless database and the view can also be accessed from other services such as Power BI and Data Factory. Serverless SQL pool, with its ability to define views, can be used to create a logical data warehouse on top of a data lake storage account, which will serve as a powerful tool for data analysis and exploration.

Provisioning and configuring Spark pools

A Spark pool is an important component of Azure Synapse Analytics that allows us to perform data exploration and processing using the Apache Spark engine. Spark pools in Azure Synapse Analytics allow us to process data using programming languages such as PySpark, Scala, C#, and Spark SQL. In this recipe, we will learn how to provision and configure Spark pools in Synapse Analytics.

Getting ready

Create a Synapse Analytics workspace, as explained in the *Provisioning an Azure Synapse Analytics workspace* recipe.

How to do it...

Let's perform the following steps to provision a Spark pool in an Azure Synapse Analytics workspace:

1. Log in to `portal.azure.com` and click **All Resources**. Search for `packtadesynapse`, the Synapse Analytics workspace created in the *Provisioning an Azure Synapse Analytics workspace* recipe. Click on the workspace. Search for **Apache Spark pools** under **Analytics pools**. Click **+ New**:

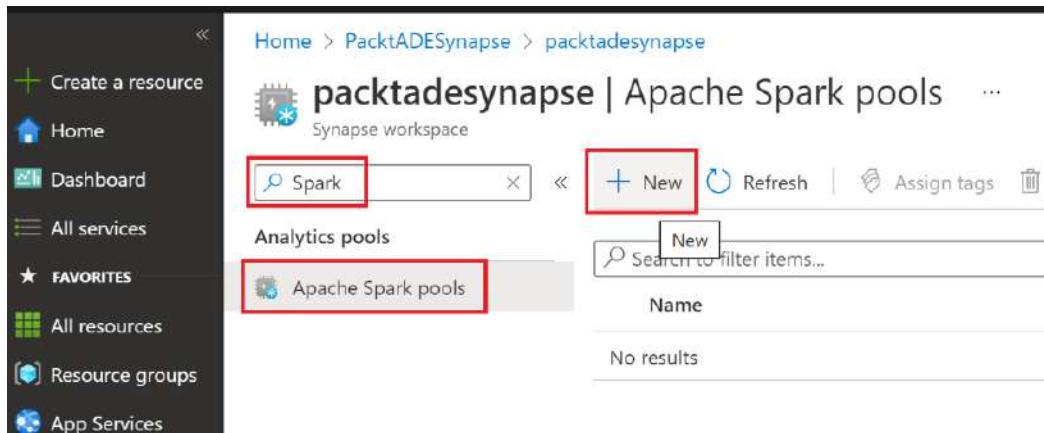


Figure 8.12 – Creating Spark pools

2. Fill in the details as follows:
 - A. Name the Spark pool `packtsparkpool`.
 - B. The **Node size family** property indicates the type of virtual machines that will be running to process the big data workload. Let's pick **Memory Optimized**, as it's typically good enough for general purpose data processing tasks. The **Hardware accelerated** type provides GPU-powered machines meant for performing heavy-duty data science and big data workloads.

- C. The **Node size** property indicates the compute and memory of virtual machines in a Spark pool. Let's pick **Small (4 vCores / 32 GB)**, as it's the cheapest option.
- D. **Autoscale** allows the Spark pool to allocate additional nodes or machines depending upon the workload. Let's leave it as **Enabled**.
- E. Set the minimum and maximum **Number of nodes** to **3** and **10**. A Spark pool can autoscale up to a maximum of 10 machines/nodes. Click on **Next: Additional settings >**:

Home > PacktADESynapse > packtadesynapse >

New Apache Spark pool

[...](#)

[* Basics](#) [* Additional settings](#) [Tags](#) [Review + create](#)

Create a Synapse Analytics Apache Spark pool with your preferred configurations. Complete the Basics tab then go to Review + create to provision with smart defaults, or visit each tab to customize.

Apache Spark pool details

Name your Apache Spark pool and choose its initial settings.

Apache Spark pool name *** packsparkpool**

Isolated compute Enabled Disabled

Node size family **Memory Optimized**

Node size **Small (4 vCores / 32 GB)**

Autoscale *** Enabled**

Number of nodes **3** **10**

Dynamically allocate executors Enabled Disabled

Estimated price **1.65 to 5.50 USD**

[View pricing details](#)

⚠️ Contact an **Owner** of the storage account, and verify that the following role assignments have been made:

- Assign the workspace MSI to the **Storage Blob Data Contributor** role on the storage account
- Assign you and other users to the **Storage Blob Data Contributor** role on the storage account

Once those assignments are made, the following Spark features can be used:
 (1) Spark Library Management, (2) Read and Write data to SQL pool databases via the Spark SQL connector, and (3) Create Spark databases and tables

[Review + create](#) [< Previous](#) [Next: Additional settings >](#)

Figure 8.13 – Creating Spark pools

- F. Leave **Automatic pausing** as **Enabled**. Automatic pausing stops the Spark pool when there are no jobs being processed.

- G. Set **Number of minutes idle** to **10**. This ensures that the pool is paused if no job is running for 10 minutes.
- H. Set the **Apache Spark** version to the latest one (**3.1** here, as of March 2022), as it ensures that we get the latest Java, Scala, .NET, and Delta Lake versions. Click **Review + create**:

Home > PacktADESynapse > packtadesynapse >

New Apache Spark pool ...

* Basics * Additional settings Tags Review + create

Customize additional configuration parameters including automatic pausing and component versions.

Automatic pausing

Configure automatic pausing. If enabled, the Apache Spark pool will automatically pause after the selected idle time.

Automatic pausing * ⓘ Enabled Disabled

Number of minutes idle * ✓

Component versions

Select the Apache Spark version for your Apache Spark pool.

Apache Spark *	3.1
Python	3.8
Scala	2.12.10
Java	1.8.0_282
.NET Core	3.1
.NET for Apache Spark	2.0
Delta Lake	1.0

Spark configuration

Upload a Spark configuration file to specify additional properties on the Spark pool. This will be referenced to configure Spark applications upon job submission. [Learn more](#) ⓘ

File upload

[Review + create](#) [< Previous](#) [Next: Tags >](#)

Figure 8.14 – Configuring Spark pools

How it works...

Creating a Spark pool defines the configuration for the nodes/virtual machines, which will be processing the big data workload as it arrives. Each time a new user logs in and submits a Spark job/Spark notebook to process data, an instance of Spark pool is created. An instance of Spark pool is basically a bunch of virtual machines/nodes configured as defined in the Spark pool configuration. A single user can use up to the maximum number of nodes defined in the pool. If there are multiple users connecting to the Spark pool, Synapse will create as many instances of Spark pool as the number of users. You will be billed for the number of active instances and for the time period during which they were active. Billing will stop for a particular instance if it remains idle longer than the idle time period defined in the Spark pool configuration.

Processing data using Spark pools and a lake database

Spark pools in a Synapse workspace allow us to process data and store them as tables inside a lake database. A lake database allows us to create tables using CSV files, Parquet files, or as Delta tables stored in the data lake account. Delta tables use Parquet files for storage and support `insert`, `update`, `delete`, and `merge` operations. Delta tables are stored in a columnar format, which is compressed, ideal for storing processed data and supporting analytic workloads. In this recipe, we will read a CSV file, perform basic processing, and load the data into a Delta table in a lake database.

Getting ready

Create a Synapse Analytics workspace, as explained in the *Provisioning an Azure Synapse Analytics workspace* recipe.

Create a Spark pool cluster, as explained in the *Provisioning and configuring Spark pools* recipe.

We need to upload the `covid-data.csv` file from <https://github.com/PacktPublishing/Azure-Data-Engineering-Cookbook-2nd-edition/blob/main/chapter07/covid-data.csv> to a folder named `CSV` in the data lake account attached to the Synapse Analytics workspace. To do so, follow *step 1 to step 4* in the *How to do it...* section from the *Analyzing data using serverless SQL pool* recipe.

How to do it...

After uploading the `covid-data.csv` file to the `CSV` folder in the data lake account, let's perform the following steps to process the data in the CSV file and load it into a Delta Lake table in a lake database:

1. Log in to `portal.azure.com`, click **All resources**, search for `packtadesynapse`, the Synapse Analytics workspace we created, and click on it. Click **Open Synapse Studio**.

2. Click on the data icon on the left, click the **Linked** tab, and expand the **Azure Data Lake Storage Gen2 | packtadesynapse | synapse (Primary)** container. Navigate to the **CSV** folder inside the **synapse** container, where the **covid-data.csv** file has been uploaded. Right-click on the **covid-data.csv** file, select the option **New notebook | Load to DataFrame**:

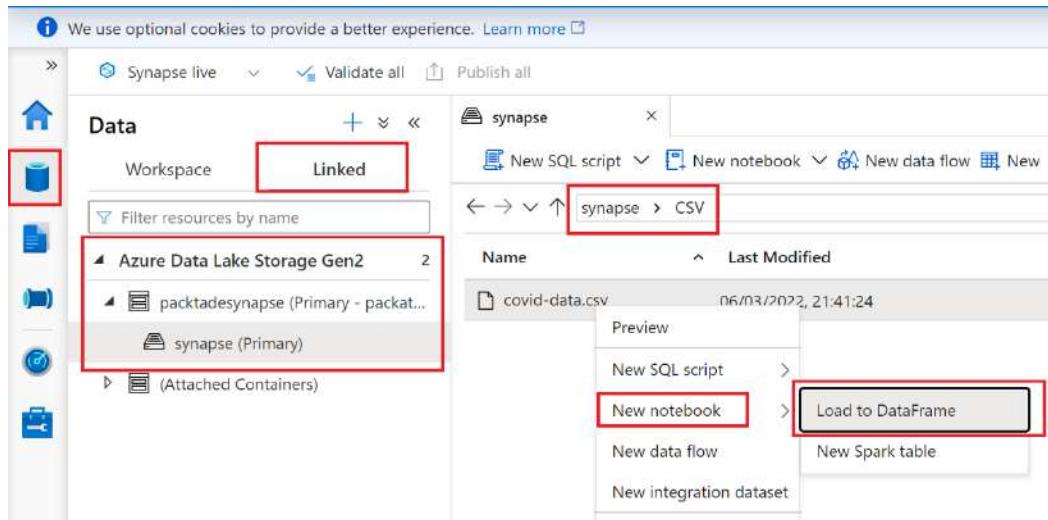


Figure 8.15 – Load to DataFrame

3. A new notebook will be created. Notebooks are used by data engineers to develop the code that will be used to process data using Synapse pools. Attach the notebook to **packtsparkpool**, the Synapse Spark pool created in the *Provisioning and configuring Spark pools* recipe in this chapter, using the **Attach to** drop-down menu at the top of the notebook. Uncomment the fourth line in the first cell by removing `##`, as our file contains a header. Hit **Run Cell** (which looks like a play button) on the left:

The screenshot shows the Azure Synapse Analytics workspace with a notebook titled 'Notebook 1' open. The 'Attached' tab is selected in the 'Data' blade. In the notebook, there is one cell containing the following PySpark code:

```

1 %%pyspark
2 df = spark.read.load('abfss://synapse@packtadesynapse.dfs.core.windows.net/CSV/covid-data.csv', format='csv'
3 ## If header exists uncomment line below
4 , header=True
5 )
6 display(df.limit(10))

```

The cell is currently executing, as indicated by the play button icon in the toolbar above the cell area, which is highlighted with a red box. The 'Attach to' dropdown menu in the toolbar is also highlighted with a red box and set to 'packtsparkpool'. The 'Language' dropdown is set to 'PySpark (Python)'. The bottom of the screen shows navigation buttons for 'Code' and 'Markdown'.

Figure 8.16 – The data loaded to the DataFrame

4. Data will be loaded to a DataFrame called `df` using the automatic PySpark code generated. Let's use the Spark SQL language to understand the data. To interact with the data using Spark SQL, we need to create a temporary view. The `createOrReplaceTempView` command helps to create a temporary view that will be visible only within the notebook. Add a new cell by hitting the **+ Code** button, paste the following command, and run the new cell:

```
df.createOrReplaceTempView("v1")
```

The output is displayed in the following screenshot:

A screenshot of a Jupyter Notebook interface. At the top, there are four tabs: 'CPV', 'Africa', 'Cape Verde', and '18/2/2021'. Below these tabs is a scroll bar. In the center, there is a code cell containing the command `df.createOrReplaceTempView("v1")`. This cell is highlighted with a red border. Above the code cell are two buttons: '+ Code' and '+ Markdown', with '+ Code' also having a red border. Below the code cell is a status bar with the text 'Press shift + enter to execute cells'.

Figure 8.17 – Creating a temporary view

5. To check out which columns are present in the `covid-data.csv` file, let's use the `Describe` Spark SQL command to list the columns in the view. Add a new cell and paste the following command. The `%%sql` command switches the programming language from PySpark to Spark SQL. We will notice that the view contains several columns (use the scroll bar to check out all of them). All the columns are also of the `string` data type:

```
%%sql
Describe v1;
```

The output is as follows:

A screenshot of a Jupyter Notebook cell showing the results of a `%%sql` command. The cell content is as follows:

```
1 %%sql
2 Describe v1;
3 ✓ 9 sec - Command executed in 8 sec 915 ms by arr.nagaraj on 7:18:26 PM, 3/19/22
```

The results are displayed in a table format:

col_name	data_type	comment
iso_code	string	null
continent	string	null
location	string	null
date	string	null
total_cases	string	null
new_cases	string	null

Figure 8.18 – The list columns

6. Let's focus on the following key columns – date, continent, location, new_cases, and new_deaths. Let's also change the data type of new_cases and new_deaths to integer and load it into a Delta table. To load it into a Delta table, we need to create a lake database first, create the new Delta table, and then load the data. The Create database command creates the database, and Create table <tablename> using Delta as <Select statement> creates the table and loads the data. Copy the following command to a new cell and run the new cell:

```
%%sql
Create database sparksql;
Create or replace table sparksql.covid
USING Delta
AS
Select date, continent,location, CAST(new_cases as int)
as new_cases,
CAST(new_deaths as int) as new_deaths from v1
```

The output of the preceding query is as follows:

▶

```
1 %%sql
2 Create database sparksql;
3 Create or replace table sparksql.covid
4 USING Delta
5 AS
6 Select date, continent,location, CAST(new_cases as int) as new_cases,
7 CAST(new_deaths as int) as new_deaths from v1
[4] ✓ 24 sec - Command executed in 1 ms by arr.nagaraj on 7:34:07 PM, 3/19/22
```

... No data available

Figure 8.19 – Creating a Delta table

7. In the previous step, we created a lake database called sparksql and a Delta table inside it called covid. Using the delta option in the CREATE or REPLACE TABLE command ensured that the table was created as a Delta table. The CAST function in the SELECT statement changed the column data type to INTEGER. Verify the data-type change using the DESCRIBE command:

```
%%sql
Describe table sparksql.covid;
```

The result of the query execution is demonstrated in the following screenshot:

```
1 %%sql
2 Describe table sparksqldb.covid;
```

✓ 1 sec - Command executed in 1 sec 61 ms by arr.nagaraj on 7:39:41

View Table Chart Export results

col_name	data_type
date	string
continent	string
location	string
new_cases	int
new_deaths	int

Figure 8.20 – The Delta table structure

8. Let's delete the rows that have NULL values in the continent column. Add a new cell and copy-paste the following command:

```
%%sql
Delete from sparksqldb.covid where continent is NULL
```

The output of the preceding query is as follows:

Code Markdown

```
▷
1 %%sql
2 Delete from sparksqldb.covid where continent is NULL
[6] ✓ 14 sec - Command executed in 14 sec 145 ms by arr.nagaraj on 7:48:18 PM, 3/19/22
```

Job execution Succeeded Spark 2 executors 8 cores

Figure 8.21 – Using a Delete statement on a Delta table

9. Delta tables have a feature called time travel, which lets us explore the previous versions of the table. We will use time travel to query the deleted rows (rows with NULL values in the continent column). To perform that as a first step, we need to find the location where the Delta table is stored. The `DESCRIBE DETAIL` command will provide a column called `location`, which will contain the location of the Delta table. Add a new cell, copy the following command, and run the cell. Copy the contents of the `location` column. Ensure to expand the column by dragging the slider to your right and copying the full path of the Delta table:

```
%%sql
DESCRIBE DETAIL sparksqlDB.covid
```

The output of the preceding query is as follows:

	format	id	name	description	location
[12]	delta	c676620d-b511-4772-abaa-4ead...	sparksqlDB.covid	null	abfss://synapse@packtadesynapse.dfs.core.windows.net/synapse/workspaces/packtadesynapse/warehouse/sparksqlDB.db/covid

Figure 8.22 – Get the Delta table location

10. On the copied location, remove the text that starts with `abfss` and goes up to `windows.net`. We only need the path that starts from the container name (`synapse`), not the storage account or protocol details. For example, if your copied location is `abfss://synapse@packtadesynapse.dfs.core.windows.net/synapse/workspaces/packtadesynapse/warehouse/sparksqlDB.db/covid`, remove `abfss://synapse@packtadesynapse.dfs.core.windows.net/` and retain `/synapse/workspaces/packtadesynapse/warehouse/sparksqlDB.db/covid`.
11. Add a new cell and copy the following Spark command. Paste the edited location path to the `load` function. The `Option("versionAsOf", 0)` function makes the command read the older version of the table. The second parameter in the `option` function indicates the version number to be read. Version number 0 would be the most recent previous version of the table, version number 1 would be the next version older than version 0, and so on. The command reads the older data to a DataFrame, which we load to a view called `old_Data`:

```
df2 = spark.read.format("delta").option("versionAsOf", 0).load("/synapse/workspaces/packtadesynapse/warehouse/sparksqlDB.db/covid")
df2.createOrReplaceTempView("old_Data")
```

The result of the query execution is demonstrated in the following screenshot:

```
[15] 1 df2 = spark.read.format("delta").option("versionAsOf", 0).load("/synapse/worksheets/packtadesynapse/warehouse/sparksqldb.db/covid")
2 df2.createOrReplaceTempView("old_Data")
✓ 4 sec - Command executed in 4 sec 15 ms by arr.nagaraj on 8:35:28 PM, 3/19/22
> Job execution Succeeded  Spark 2 executors 8 cores
View in monitoring  Open Spark UI
DataFrame[date: string, continent: string, location: string, new_cases: int, new_deaths: int]
```

Figure 8.23 – Reading an old version of the Delta table

12. Execute a SELECT statement against the `old_Data` view to check out the rows that were deleted. Add a new cell, copy the following command, and execute the new cell. We will be able to read the deleted rows using the time travel feature on the Delta table:

```
%%sql
SELECT * FROM old_Data WHERE continent IS NULL
```

The result of the query execution is demonstrated in the following screenshot:

```
1 %%sql
2 Select * from old_Data where continent is NULL
✓ 6 sec - Command executed in 5 sec 801 ms by arr.nagaraj on 9:25:55 PM, 3/19/22
> Job execution Succeeded  Spark 2 executors 8 cores
View in monitoring  Open Spark UI
```

View Table Chart I → Export results ↻

date	continent	location	new_cases	new_deaths
1/1/2021	null	South America	45902	1080
2/1/2021	null	South America	38823	875
3/1/2021	null	South America	39551	892
4/1/2021	null	South America	50539	1199

Figure 8.24 – SELECT statement on Delta table

How it works...

The Spark pools in a Synapse workspace allow us to seamlessly load CSV files to Delta tables using notebooks. Notebooks allow us to effortlessly switch between PySpark and SQL. Delta tables support data manipulation commands such as `update`, `delete`, and `merge`, and capabilities such as time travel make it very efficient for data processing tasks in data engineering projects.

Querying the data in a lake database from serverless SQL pool

Lake databases are created from Synapse Spark pools and typically consist of Delta tables. The following recipe will showcase how we could read the data stored in Delta tables from serverless SQL pool.

Getting ready

Create a Synapse Analytics workspace, as explained in the *Provisioning an Azure Synapse Analytics workspace* recipe in this chapter.

Create a Spark pool, as explained in the *Provisioning and configuring Spark pools* recipe in this chapter.

Create a lake database and Delta table, as explained in the *Processing data using Spark pools and lake database* recipe in this chapter.

How to do it...

Perform the following steps to query the data:

1. Log in to `portal.azure.com`, click **All Resources**, search for **packtadesynapse**, the Synapse Analytics workspace that we created, and click on it. Click **Open Synapse Studio**. Click on the data icon on the left, click the **Linked** tab, and expand the **Azure Data Lake Storage Gen2 | packtadesynapse | synapse (Primary)** container. Navigate to the following folder path: `/synapse/workspaces/packtadesynapse/warehouse/sparksqldb.db`. The usual path structure is `<SynapseContainerName>/synapse/workspaces/<WorkspaceName>/warehouse/<lakedatabasename.db>`. So, if you have named your lake database or table name differently, then it will vary from mine here. You will find a folder with the Delta table name (**covid**). Right-click on it and select **New SQL script | Create external table**:

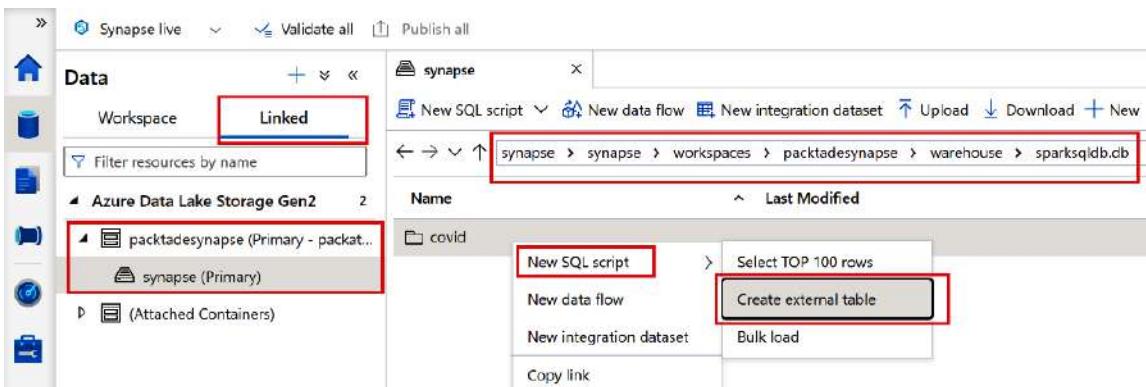


Figure 8.25 – Create external table

2. Synapse will detect the file type and schema. Hit **Continue** to generate the external table creation script:

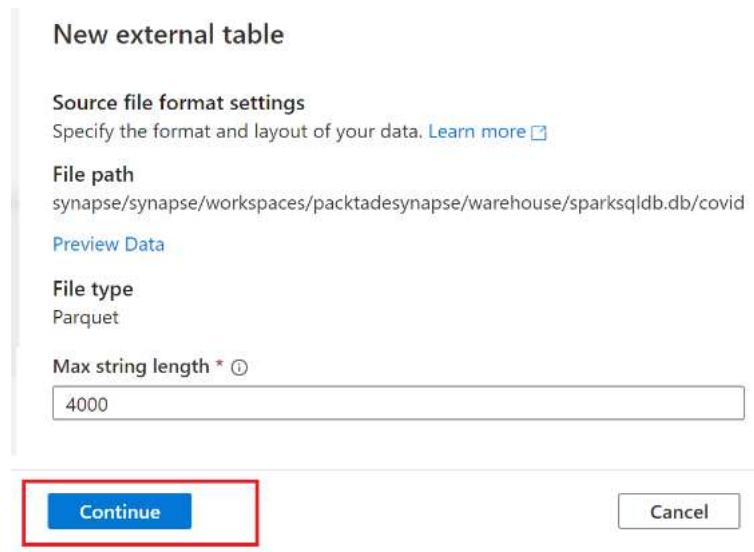


Figure 8.26 – Generation of the external table script

3. Leave the **Select SQL pool** option as **Built-in**. **Built-in** is the in-built serverless SQL pool. Select + New to create a new serverless database:

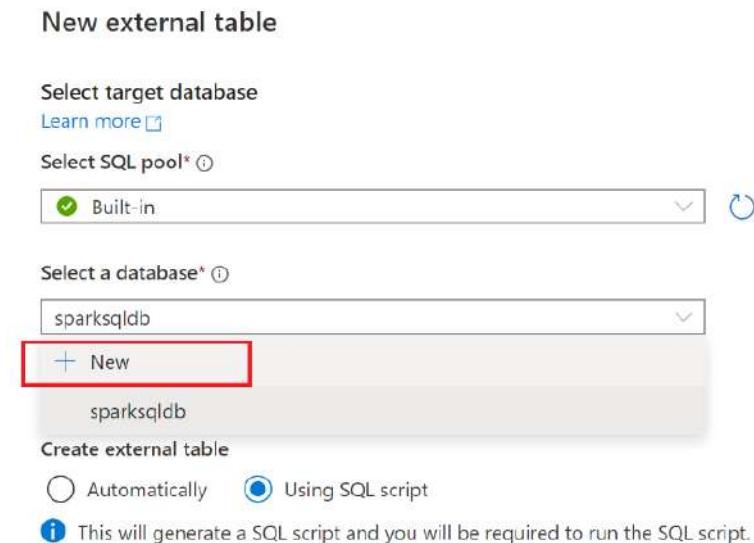


Figure 8.27 – A new serverless database

4. Name the database ServerlessSQLdb and click the **Create** button:

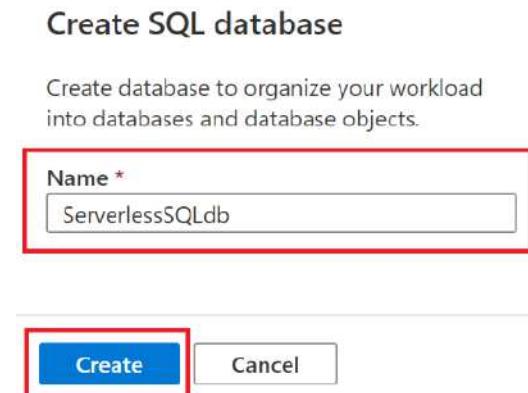


Figure 8.28 – Create a serverless database

5. Name the external table dbo.covid_ext. Click **Open script**:

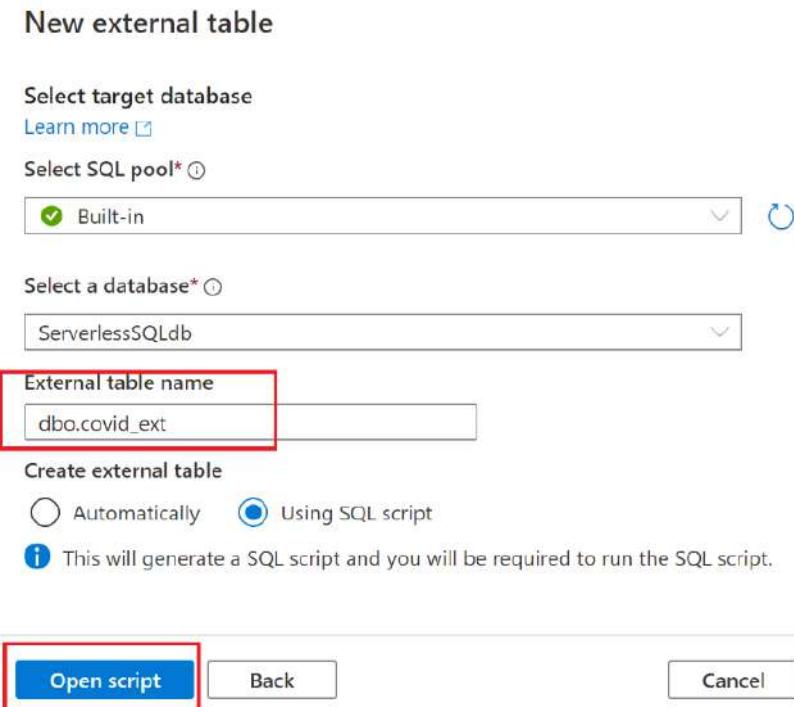
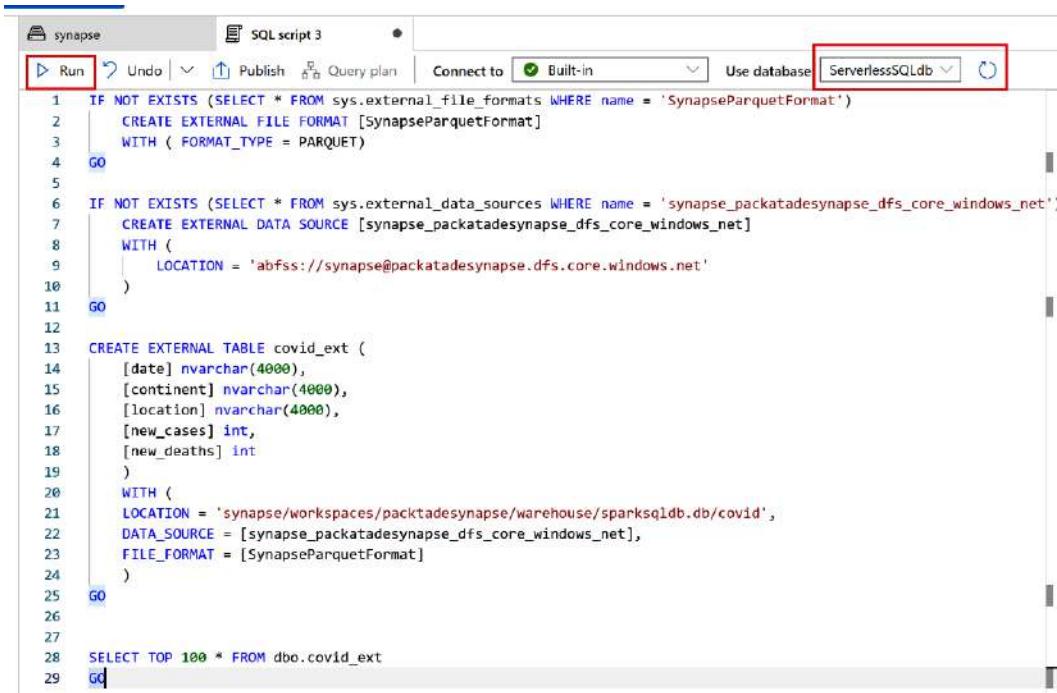


Figure 8.29 – Creating an external table

6. To create an external table in Synapse SQL pool, we need to create the following objects: an **external file format** and an **external data source** first. Using Synapse Studio, we can generate the script to create the **external file format**, **external data source**, and **external table**. Select **ServerlessSQLdb** from the **Use database** dropdown and click the **Run** button to create the external table:



The screenshot shows the Synapse Studio interface with the following details:

- Toolbar: Run, Undo, Publish, Query plan, Connect to (Built-in), Use database (ServerlessSQLdb), Refresh.
- Code Editor: A SQL script titled "SQL script 3" containing T-SQL code to create an external file format, external data source, and an external table named "covid_ext".
- Script Content:

```
1 IF NOT EXISTS (SELECT * FROM sys.external_file_formats WHERE name = 'SynapseParquetFormat')
2     CREATE EXTERNAL FILE FORMAT [SynapseParquetFormat]
3     WITH (FORMAT_TYPE = PARQUET)
4 GO
5
6 IF NOT EXISTS (SELECT * FROM sys.external_data_sources WHERE name = 'synapse_packatadesynapse_dfs_core_windows_net')
7     CREATE EXTERNAL DATA SOURCE [synapse_packatadesynapse_dfs_core_windows_net]
8     WITH (
9         LOCATION = 'abfss://synapse@packatadesynapse.dfs.core.windows.net'
10    )
11 GO
12
13 CREATE EXTERNAL TABLE covid_ext (
14     [date] nvarchar(4000),
15     [continent] nvarchar(4000),
16     [location] nvarchar(4000),
17     [new_cases] int,
18     [new_deaths] int
19 )
20 WITH (
21     LOCATION = 'synapse/workspaces/packatadesynapse/warehouse/sparksqldb.db/covid',
22     DATA_SOURCE = [synapse_packatadesynapse_dfs_core_windows_net],
23     FILE_FORMAT = [SynapseParquetFormat]
24 )
25 GO
26
27
28 SELECT TOP 100 * FROM dbo.covid_ext
29 GO
```

Figure 8.30 – Creation of the external table

7. Upon clicking the **Run** button, we will be able to see the data read from the external table successfully:

```

28  SELECT TOP 100 * FROM dbo.covid_ext
29  GO

```

Results Messages ^

View Table Chart Export results ▾

Search

date	continent	location	new_cases	new_deaths
29/12/2021	Africa	South Africa	9020	81
30/12/2021	Africa	South Africa	12978	126
31/12/2021	Africa	South Africa	11754	84
1/1/2022	Africa	South Africa	9793	53

Figure 8.31 – Reading the external table

How it works...

To access the Delta table in a lake database, we need to create an external table in serverless SQL pool. We identified the folder where the Delta table was stored and we created an external table against it in serverless SQL pool. The external table acts as a link to the files stored in the Delta table. While the files reside in the Delta table, files appear as a table to end users in Serverless SQL pool. So, when a user queries the external table using a T-SQL script in serverless SQL pool, it will seamlessly read from the Delta table's files and present it in a tabular format.

The lake database created a folder for each Delta table. So, it made it easier for us to create an external table against the folder of the Delta Lake table, which implies that we can seamlessly query the data from lake database Delta table in a serverless SQL pool database via external table. Changes to the Delta table are handled by adding or removing Parquet files inside the Delta Lake table folder using the Apache Spark engine. As we have created the external table against the table's folder (not against any specific file), all the changes happening in the Delta Lake table will immediately be reflected in the serverless SQL pool's external table without any additional effort.

Scheduling notebooks to process data incrementally

Consider the following scenario. Data is loaded daily into the data lake in the form of CSV files. The task is to create a scheduled batch job that processes the files loaded daily, performs basic checks, and loads the data into the Delta table in the lake database. This recipe addresses this scenario by covering the following tasks:

1. Only reading the new CSV files that are loaded to the data lake daily using Spark pools and notebooks

2. Processing and performing upserts (update if the row exists, insert if it doesn't), and loading data into the Delta lake table using notebooks
3. Scheduling the notebook to operationalize the solution

Getting ready

Create a Synapse Analytics workspace, as explained in the *Provisioning an Azure Synapse Analytics workspace* recipe in this chapter.

Create a Spark pool, as explained in the *Provisioning and configuring Spark pools* recipe in this chapter.

Download the `TransDtls-2022-03-20.csv` file from <https://github.com/PacktPublishing/Azure-Data-Engineering-Cookbook-2nd-edition/blob/main/chapter08/TransDtls-2022-03-20.csv>. Create a folder called `transaction-data` inside the `synapse` container in the `packtadesynapse` data lake account. You can use Synapse Studio's data pane to do the same. For detailed instructions on creating a folder and manually uploading files to Synapse Studio, refer to *step 1 to step 4* in the *How to do it...* section of the *Analyzing data using serverless SQL Pool* recipe. Upload the file, as shown in the following picture:

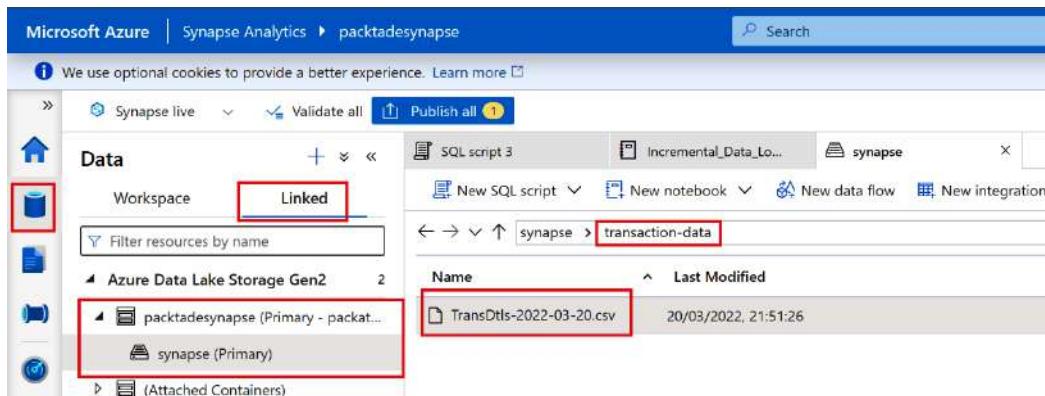


Figure 8.32 – Uploading the file

How to do it...

In this scenario, the `TransDtls-2022-03-20.csv` file contains the data about transactions that have occurred in a store. Let's assume that the file has the loading date suffixed to it. So, a file that was loaded to the data lake on March 20th will be named `TransDtls-2022-03-20.csv`, a load from March 21st will be named `TransDtls-2022-03-21.csv`, and so on. Our notebook and scheduled task should read only the latest file (and not all the files in the `transaction-data` folder), so that it can process the data incrementally. To process the data incrementally and load the data to a Delta Lake, let's perform the following steps:

1. Click the **Develop** icon on the left-hand side of Synapse Studio. In the **Notebooks** section, click on the three dots and select **New notebook**:

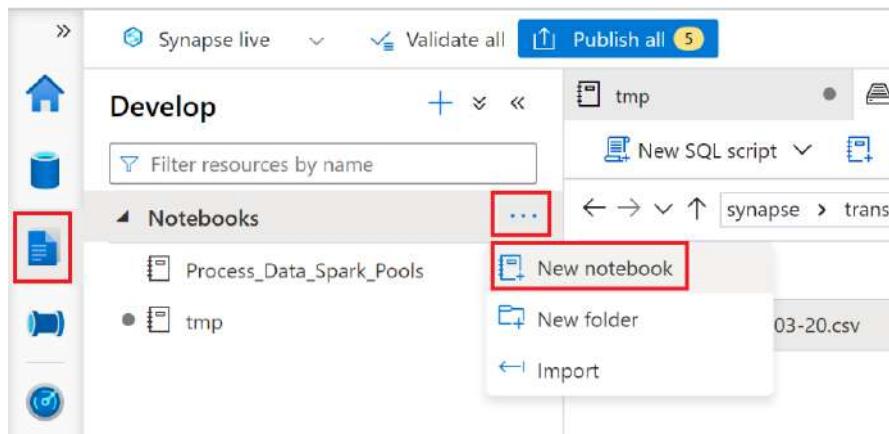


Figure 8.33 – Select New notebook

2. Name the notebook `Incremental_Data_Load` by typing the name into the **Properties** section on the left. Attach the notebook to the `packtsparkpool` cluster using the **Attach to** drop-down option at the top. Copy the following Scala script and paste it into the first cell in the notebook to only read the latest file from the `transaction-data` folder into a DataFrame. The `Java.time.LocalDate.now` command gets the current date and we use the current date to construct the name of the file to be read. This way, even if there are hundreds of files in the folder, the notebook will only read the latest file. Hit the **Run** button (which looks like a play button) on the left. The latest file is loaded to the DataFrame and named `transaction_today`:

```
%%spark
val date = java.time.LocalDate.now
val transaction_today = spark.read.format("csv") .
  option("header", "true").option("inferSchema", "true") .
  load("/transaction-data/TransDtls-" + date + ".csv")
display(transaction_today)
```

The output of the preceding query is displayed in the following screenshot:

The screenshot shows the Azure Data Studio interface. On the left, there's a sidebar with icons for Home, Databases, Notebooks, and tmp. Under Notebooks, there are two entries: 'Process_Data_Spark_Pools' and 'tmp'. The main area shows a notebook titled 'Incremental_Data_Load'. The 'Attach to' dropdown at the top is set to 'packtsparkpool'. The notebook content consists of a single code cell with the following PySpark code:

```

1 %%spark
2 val date = java.time.LocalDate.now
3 val transaction_today = spark.read.format("csv").option("header", "true").option("inferSchema", "true").load
4 display(transaction_today)

```

The output of the cell shows the command was executed in 5 seconds with 390 ms by arr.nagaraj on 11:00:26 PM, 3/20/22. It indicates a 'Job execution Succeeded' with Spark 2 executors 8 cores. Below the cell, there are tabs for View, Table, Chart, and Export results. The Table tab is selected, showing a DataFrame with four columns: Transaction_id, Order_id, Order_dt, and customer_id. The data is as follows:

Transaction_id	Order_id	Order_dt	customer_id
1	1	2022-03-10	C-1
2	1	2022-03-10	C-1
3	1	2022-03-10	C-1

To the right of the notebook content, there's a 'Properties' panel. It shows the 'Name' is 'Incremental_Data_Load', 'Type' is 'ipynb notebook', 'Size' is 23,766 bytes, and 'Notebook settings' include 'Include cell output when saving' checked.

Figure 8.34 – Reading the latest file

- Add a new cell using the + **Code** button and copy-paste the command to create a temporary view using the DataFrame. Hit the **Run** button:

```

%%spark
transaction_today.createOrReplaceTempView("transaction_today")

```

The output of the query execution is demonstrated in the following screenshot:

The screenshot shows a new code cell being added. The '+' button for 'Code' is highlighted with a red box. The cell content is:

```

1 %%spark
2 transaction_today.createOrReplaceTempView("transaction_today")

```

The output of the cell shows the command was executed in 1 sec 792 ms by arr.nagaraj on 11:13:23 PM, 3/20/22.

Figure 8.35 – Reading the DataFrame

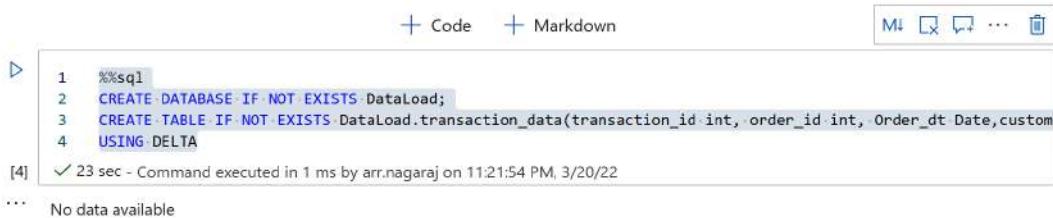
- Add a new cell and use the following SQL script to create a lake database called **DataLoad** and a Delta table called **transaction_Data**. The **If not exists** clause in the **create** statements ensures that the database and the table are only created the first time that the notebook is run. Hit the **Run** button to create the database and table:

```

%%sql
CREATE DATABASE IF NOT EXISTS DataLoad;
CREATE TABLE IF NOT EXISTS DataLoad.transaction_
data(transaction_id int, order_id int, Order_
dt Date, customer_id varchar(100), product_id
varchar(100), quantity int, cost int)
USING DELTA

```

The output is as follows:



A screenshot of an Azure Synapse Analytics workspace. At the top, there are tabs for 'Code' and 'Markdown'. On the right, there are icons for 'Copy', 'Edit', and 'More'. Below the tabs, a command is shown in a code editor:

```

1 %%sql
2 CREATE DATABASE IF NOT EXISTS DataLoad;
3 CREATE TABLE IF NOT EXISTS DataLoad.transaction_data(transaction_id int, order_id int, Order_dt Date,custom
4 USING DELTA

```

[4] ✓ 23 sec - Command executed in 1 ms by arr.nagaraj on 11:21:54 PM, 3/20/22

... No data available

Figure 8.36 – Table creation

5. Add a new cell and copy the following script to **upsert** data into the table. If the latest file in the `transaction-data` folder contains information about transactions that already exist in the Delta table, then the Delta table needs to be updated with the values from the latest file, and if the latest file contains transactions that don't exist in the Delta table, then they need to be inserted. The script uses the `merge` command, which performs the following tasks to achieve the update/insert commands:
 - A. Compares `transaction_id` on the latest file in the `transaction-data` folder (using the `transaction_today` view's `transaction_id` column) with the `transaction_data` table's (as in, the Delta table's) `transaction_id` column to see whether the file contains data about older transactions or new transactions. The comparison is carried out using the `merge` statement's `ON` clause.
 - B. If `transaction_id` from the file already exists in the `transaction_data` table, it implies that the file contains rows about older transactions, and hence, it updates all the columns of the `transaction_data` table with the latest data from the file. The `WHEN Matched` clause in the `merge` statement helps to achieve this.
 - C. If `transaction_id` doesn't exist in the Delta table but it does exist in the file, it is a new transaction and it is therefore inserted into the table. The `WHEN NOT Matched` clause in the `merge` statement helps to achieve this.
 - D. On the `WHEN Matched` clause, additional `NULL` checks are added, using the `is not null` clause to ensure that invalid rows are not inserted into the table:

```

%%sql
Merge into DataLoad.transaction_data source
Using transaction_today target on source.transaction_id =
target.transaction_id
WHEN MATCHED THEN UPDATE SET *
WHEN NOT MATCHED AND (target.transaction_id is not null
or target.order_id is not null or target.customer_id is
not null)
THEN INSERT *

```

The output of the query is as follows:

```

1 %%sql
2 Merge into DataLoad.transaction_data_source
3 Using transaction_today target on source.transaction_id = target.transaction_id
4 WHEN MATCHED THEN UPDATE SET *
5 WHEN NOT MATCHED AND (target.transaction_id is not null or target.order_id is not null or target.customer_id is not null)
6 THEN INSERT *.
7
[9] ✓ 18 sec - Command executed in 18 sec 198 ms by arr.nagaraj on 11:49:47 PM, 3/20/22
> Job execution Succeeded  Spark 2 executors 8 cores
View in monitoring  Open Spark UI

```

Figure 8.37 – A merge statement to upsert data

6. Hit the **Publish** button at the top to save the notebook:

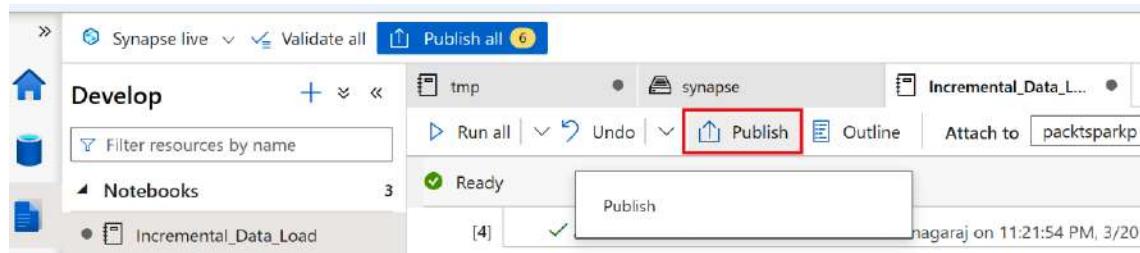


Figure 8.38 – Publishing the notebook

7. To schedule the notebook to run daily, we need to add the notebook to a pipeline. Hit the add to pipeline button in the top-right corner and select **New pipeline**:

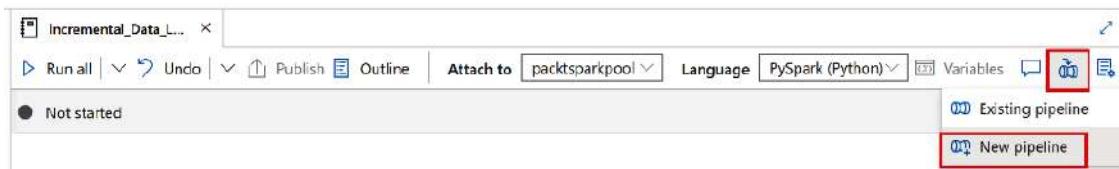


Figure 8.39 – Adding a notebook to a pipeline

8. Name the pipeline `Incremental_Data_Load`. Publish it by clicking the **Publish** button. Click **Add trigger** and select **New/Edit**:

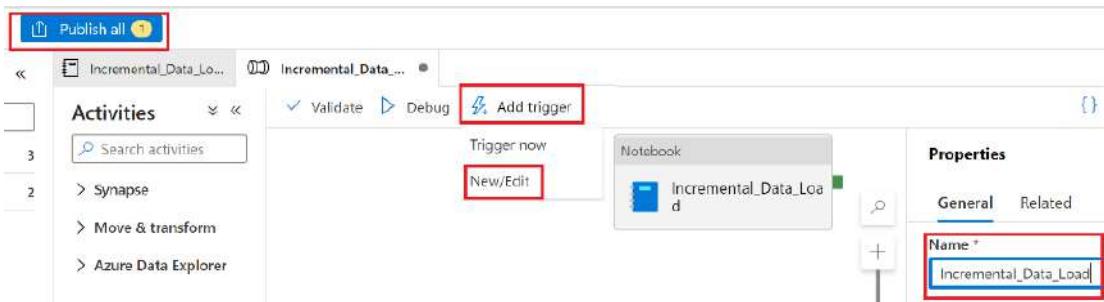


Figure 8.40 – Adding a schedule to the notebook

9. Click + New under Add triggers:

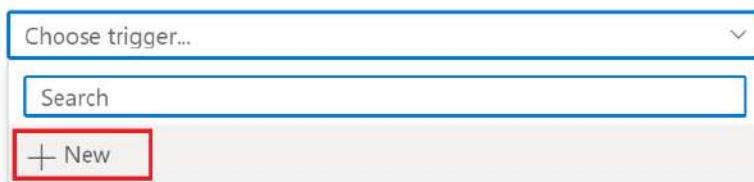
Add triggers

Figure 8.41 – Adding a new trigger

Name the trigger `Incremental_Data_Load`. Under **Recurrence**, set the schedule to run every 1 day. Under **Execute at these times**, type in 9 for **Hours** and 0 for **Minutes**. Click **OK** to schedule

New trigger

Name *

Description

Type *

Start date *

Time zone *

Recurrence *

Execute at these times

Schedule execution times
 Specify an end date

Annotations

Start trigger Start trigger on creation

Figure 8.42 – Adding a trigger

10. There are no parameters to be passed. Hit **OK** and proceed:

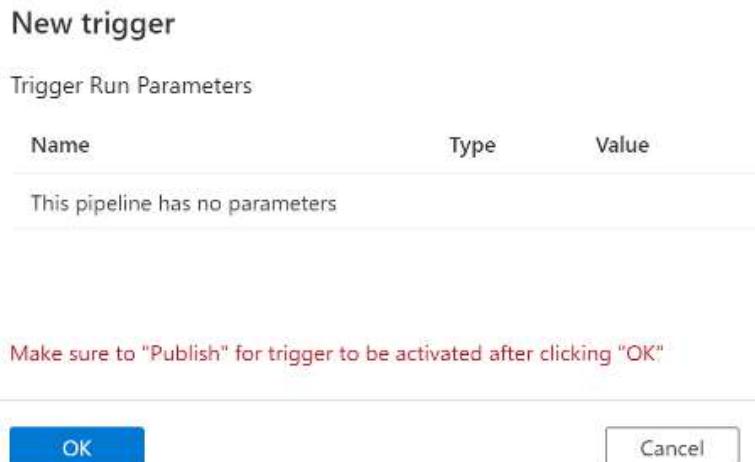


Figure 8.43 – Adding trigger run parameters

11. Hit the **Publish all** button to finish scheduling the notebook via a pipeline for a daily run:



Figure 8.44 – Publishing the trigger

How it works...

Processing data incrementally is a common scenario within data engineering projects. Synapse notebooks are extremely powerful and can be used to identify the new files to be loaded, process them alone, and load them into a Delta table. The MERGE statement is very effective at identifying the new or old transaction records and performing insert/update on the Delta table accordingly. The notebook was added to a pipeline at the click of a button and the pipeline was scheduled to run daily to process the files that are loaded every day. The processed Delta table, which is the outcome of this recipe, is typically consumed by a reporting application such as Power BI to get insights out of the processed data.

Visualizing data using Power BI by connecting to serverless SQL pool

Power BI is an excellent data visualization tool and is often used to consume the processed data in Synapse. Power BI can connect to objects (views or external tables, for example) in serverless SQL pool. In this recipe, we will create a Power BI report that will connect to an external table defined in serverless SQL pool.

Getting ready

Create a Synapse Analytics workspace, as explained in the *Provisioning an Azure Synapse Analytics workspace* recipe of this chapter.

Create a Spark pool, as explained in the *Provisioning and configuring Spark pools* recipe of this chapter.

Create a lake database and Delta table, as explained in the *Processing data using Spark pools and lake database* recipe in this chapter.

Create an external table in serverless SQL pool, as described in the *Querying the data in a lake database from serverless SQL pool* recipe.

Download the latest version of Power BI Desktop from <https://powerbi.microsoft.com/en-us/downloads/> and install Power BI Desktop on your machine.

How to do it...

Perform the following steps to connect a Delta Lake table to a Power BI report via serverless SQL pool:

1. Open Power BI Desktop and click the **Get data** button:

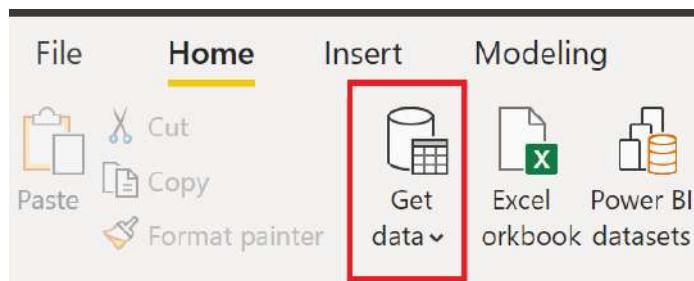


Figure 8.45 – Get data

2. Search for Synapse, select **Azure Synapse Analytics SQL**, and click the **Connect** button:

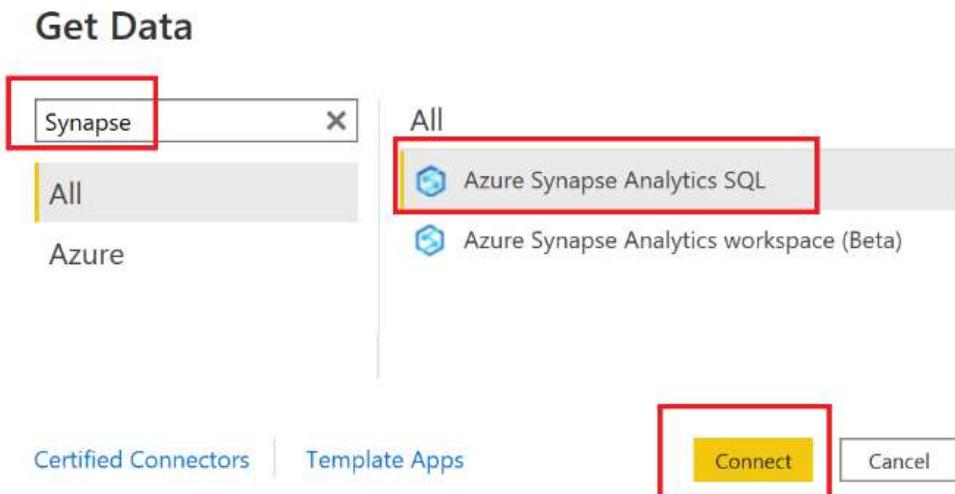


Figure 8.46 – Connecting data

3. Log in to `portal.azure.com`, click **All Resources**, search for `packtadesynapse` (the Synapse Analytics workspace that we created), and click on it. On the **Overview** page, copy the **Serverless SQL endpoint**:

Resource group (move)	: PacktADESynapse	Networking	: Show firewall settings
Status	: Succeeded	Primary ADLS Gen2 account	: https://packtadesynapse.dfs.core.windows.net
Location	: East US	Primary ADLS Gen2 file system	: synapse
Subscription (move)	: Visual Studio Ultimate with MSDN	SQL admin username	: sqladminuser
Subscription ID	:	SQL Active Directory ad...	: live.com#arr.nagaraj@gmail.com
Managed virtual network	: Yes	Dedicated SQL endpoint	: packtadesynapse.sql.azuresynapse.net
Managed Identity object ...	:	Serverless SQL endpoint	: packtadesynapse-on-demand.sql.azuresynapse.net
Workspace web URL	: https://web.azuresynapse.net?workspace=%2bsubscript...	Development endpoint	: https://packtadesynapse.dev.azuresynapse.net
Tags (edit)	: Click here to add tags		

Figure 8.47 – The Serverless SQL endpoint

4. Paste the copied serverless SQL endpoint into the Power BI connection details prompt and click **OK**:

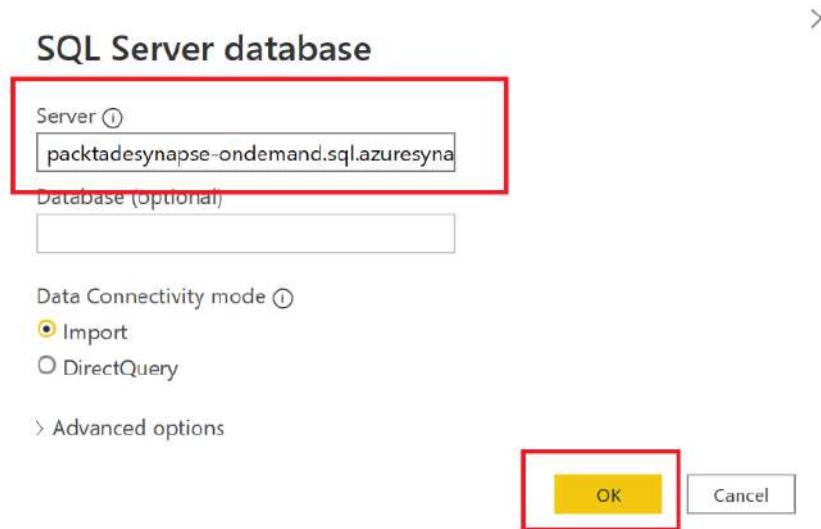


Figure 8.48 – Connect to Power BI

5. Select **Microsoft account** as the connection option. Sign in using the same account that you used to connect to `portal.azure.com`:



Figure 8.49 – Connect to Synapse

6. After signing in, click the **Connect** button:

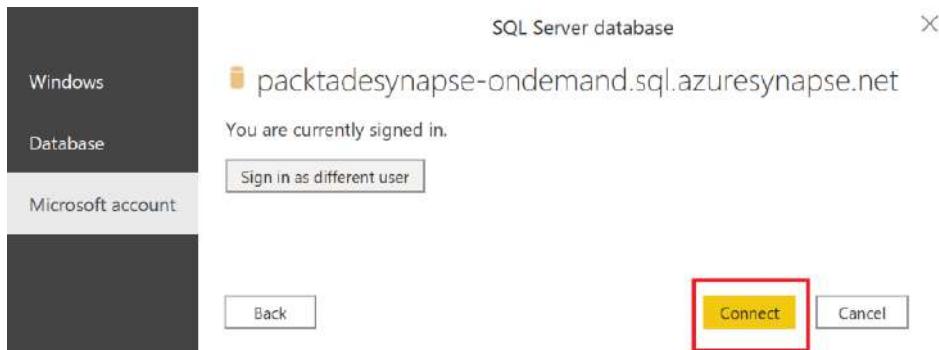


Figure 8.50 – Connect to Synapse

7. Expand **ServerlessSQLdb** and select **covid_ext**, the external table that we created in the *Querying the data in a lake database from serverless SQL pool* recipe. Click **Load**:

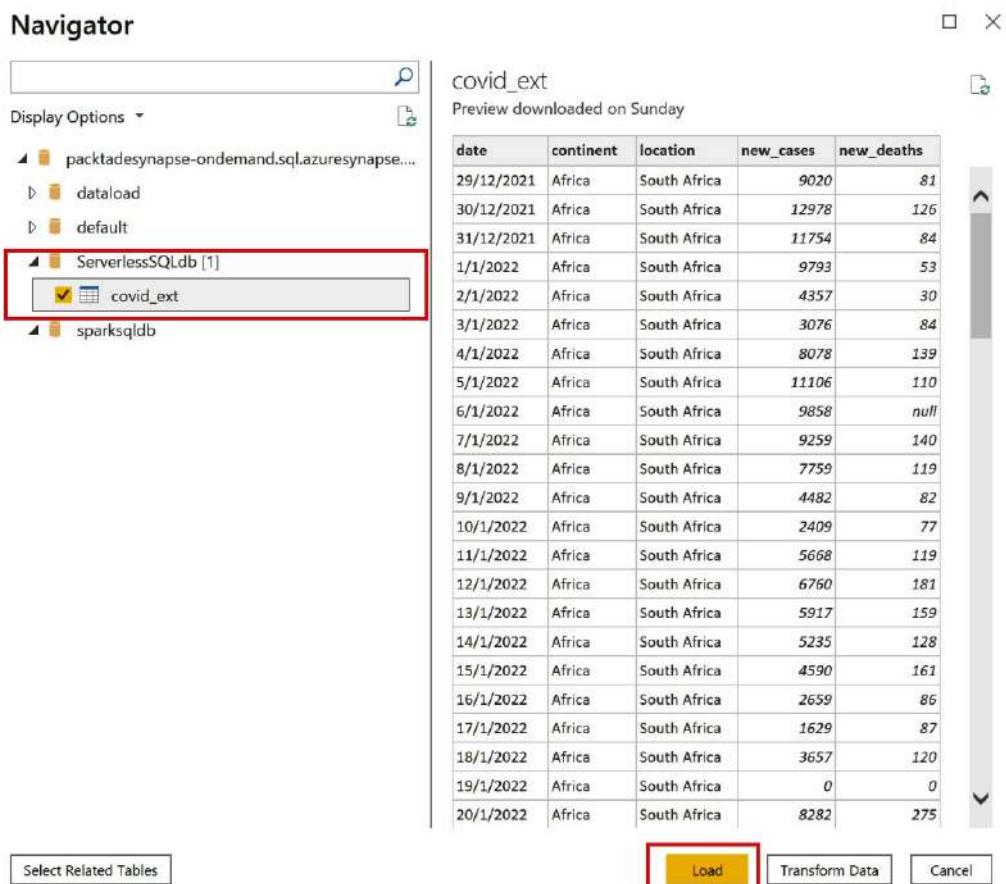


Figure 8.51 – Loading the Power BI data

8. Select the **location** column and the **new_deaths** column from the **covid_ext** table. Select the map visual icon. Place the **location** column under the **Location** property of the map visual and **new_deaths** as the **Size** property of the map visual. We are able to visualize the data effectively as follows:

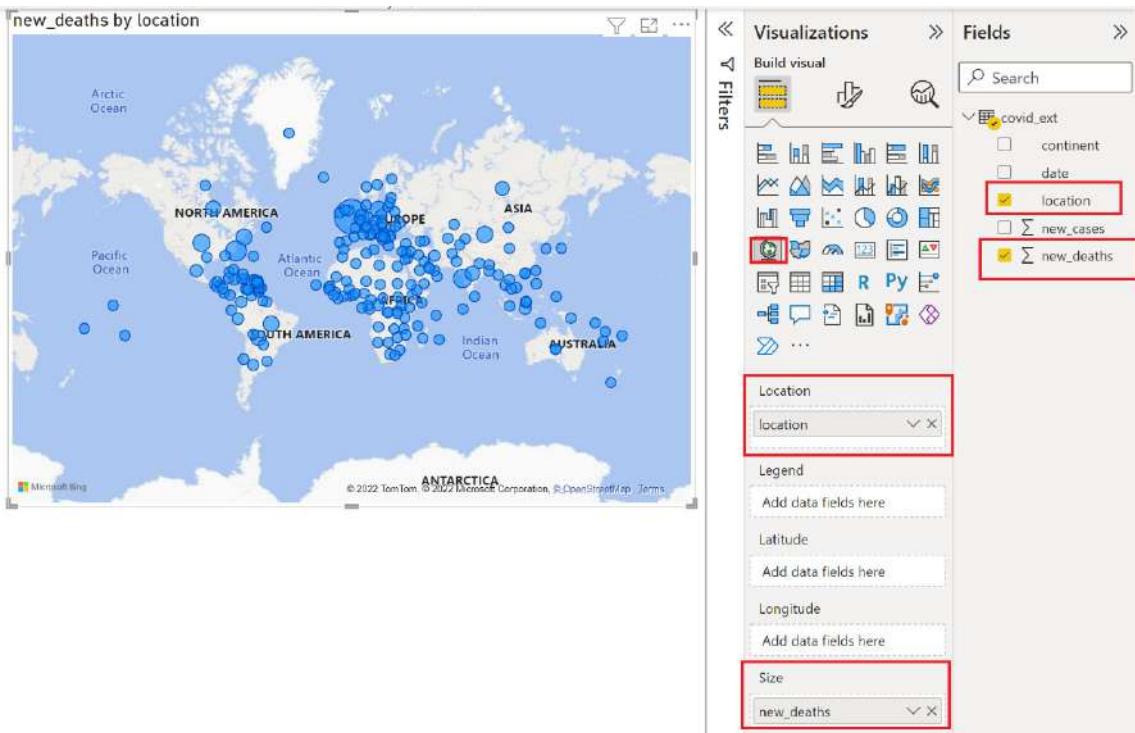


Figure 8.52 – Visualizing the data from Power BI

9. Go to the **File** menu and select the **Save** option to save the report.

How it works...

The Power BI report uses the Azure Active Directory account's context to connect to the Synapse serverless SQL pool. The serverless SQL pool's external table, which the Power BI report reads, is connected to a Delta table created in a lake database, and hence, we are able to visualize the data processed in the lake database in Power BI via a serverless SQL pool.

9

Transforming Data Using Azure Synapse Dataflows

As introduced in *Chapter 8, Processing Data Using Azure Synapse Analytics*, Azure Synapse Analytics comprises three key components – a Synapse integration pipeline to ingest and transform the data, a SQL pool and a Spark pool to process and serve the data, and Power BI integration to visualize the data. This chapter will focus on performing data transformations via Synapse data flows, a critical component of Synapse integration pipelines. Azure Synapse data flows allow us to perform transformations on the data such as converting data format, filtering, merging, and sorting while moving the data from the source to the destination. The best part about Synapse data flows is the user-friendly interface that allows us to perform complex data transformations in a few clicks.

In this recipe, we will be performing the following:

- Copying data using a Synapse data flow
- Performing data transformation using activities such as join, sort, and filter
- Monitoring data flows and pipelines
- Configuring partitions to optimize data flows
- Parameterizing mapping data flows
- Handling schema changes dynamically in data flows using schema drift

By the end of the chapter, you will have learned how to copy data to Parquet files using data flows, perform data transformation using data flows, build dynamic and resilient data flows using parameterization and schema drifting, and monitor data flows and pipelines.

Technical requirements

For this chapter, you will need the following:

- A Microsoft Azure subscription

Copying data using a Synapse data flow

In this recipe, we will convert a CSV file into the Parquet format using a Synapse data flow. We will be performing the following tasks to achieve this:

- Provisioning a Synapse Analytics workspace and a Synapse integration pipeline.
- Creating a data flow activity in the Synapse integration pipeline.
- Building the data flow activity to copy and convert the CSV file to the Parquet format.

Getting ready

To get started, do the following:

1. Log in to <https://portal.azure.com> using your Azure credentials.
2. Create a Synapse Analytics workspace, as explained in the *Provisioning an Azure Synapse Analytics workspace* recipe of *Chapter 8, Processing Data Using Azure Synapse Analytics*.
3. Download the files – `transaction_table-t1.zip` and `transaction_table-t2.zip` – from <https://github.com/PacktPublishing/Azure-Data-Engineering-Cookbook-2nd-edition/upload/main/chapter9>:
4. Unzip them. In the Synapse Analytics workspace, create a folder named `csv` (if it doesn't already exist) in the data lake account attached to the Synapse Analytics workspace.
5. Upload the files, `transaction_table-t1.csv` and `transaction_table-t2.csv`, to the `csv` folder.

To see detailed example screenshots for a similar task, follow steps 1 to 4 in the *How to do it...* section of the *Analyzing data using Serverless SQL pool* recipe from *Chapter 8, Processing Data Using Azure Synapse Analytics*.

How to do it...

Perform the following steps to copy data using a Synapse data flow:

1. Log in to `portal.azure.com`, go to **All resources**, and search for **packtadesynapse**, the Synapse Analytics workspace created in the *Provisioning an Azure Synapse Analytics workspace* recipe of *Chapter 8, Processing Data Using Azure Synapse Analytics*. Click on the workspace. Click on **Open Synapse Studio**:

The screenshot shows the Azure portal interface for a Synapse workspace named 'packtadesynapse'. The left sidebar contains navigation links: Overview (selected), Activity log, Access control (IAM), Tags, Diagnose and solve problems, Settings (with sub-links for Azure Active Directory, Properties, and Locks), Analytics pools (with sub-links for SQL pools, Apache Spark pools, and Data Explorer pools (preview)), and Security (with sub-link for Encryption). The main content area displays workspace details under the 'Essentials' tab, including Resource group (PacktADESynapse), Status (Succeeded), Location (empty), Subscription (Visual Studio Ultimate), Subscription ID (empty), Managed virtual network (Yes), Managed Identity object (empty), Workspace web URL (<https://web.azuresynapse.>), and Tags (with a link to 'Click here to add tags'). Below this is a 'Getting started' section with a red-bordered box containing a blue hexagonal icon with a white 'S' and the text 'Open Synapse Studio', followed by the instruction 'Start building your fully-integrated analytics solution and unlock new insights.' and a 'Open' button.

Figure 9.1 – Opening Synapse Studio

2. Click on the blue cylinder (the data symbol) on the left-hand side, which will take you to the **Data** section. Click on the **Linked** tab. Expand the data lake account of the Synapse workspace (**packtadesynapse** for this example). Click on the **synapse (Primary)** container. Click on the **+ New folder** button and create a folder called `transaction_table-t1-parquet`. We will use a Parquet folder as the destination to copy the Parquet file converted from the CSV file.

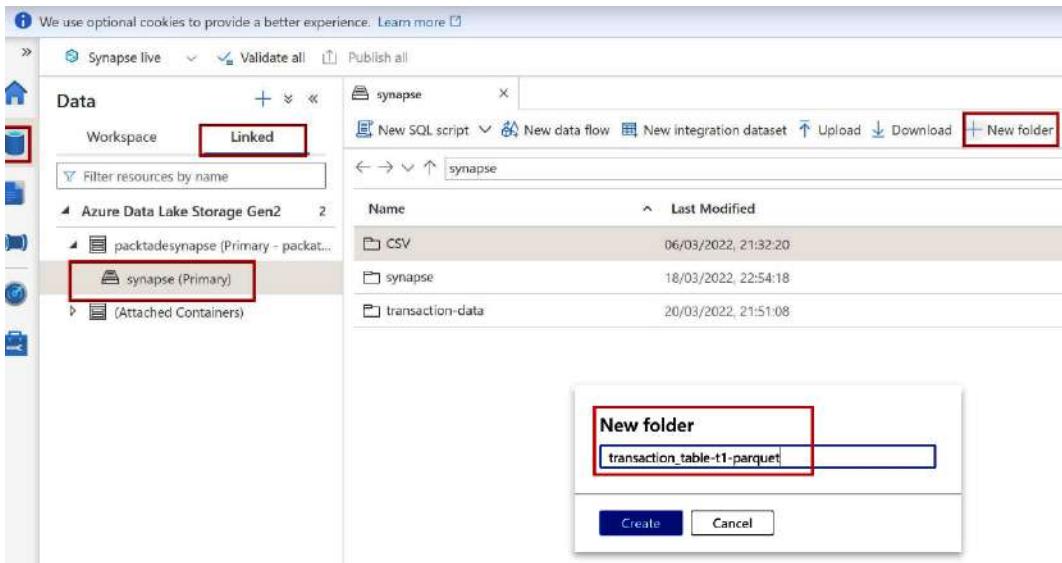


Figure 9.2 – Creating the Parquet folder

3. Click on the pipe-like icon on the left-hand side of Synapse Studio. It will open the integration pipeline development area. Hit the + button and click on **Pipeline**:

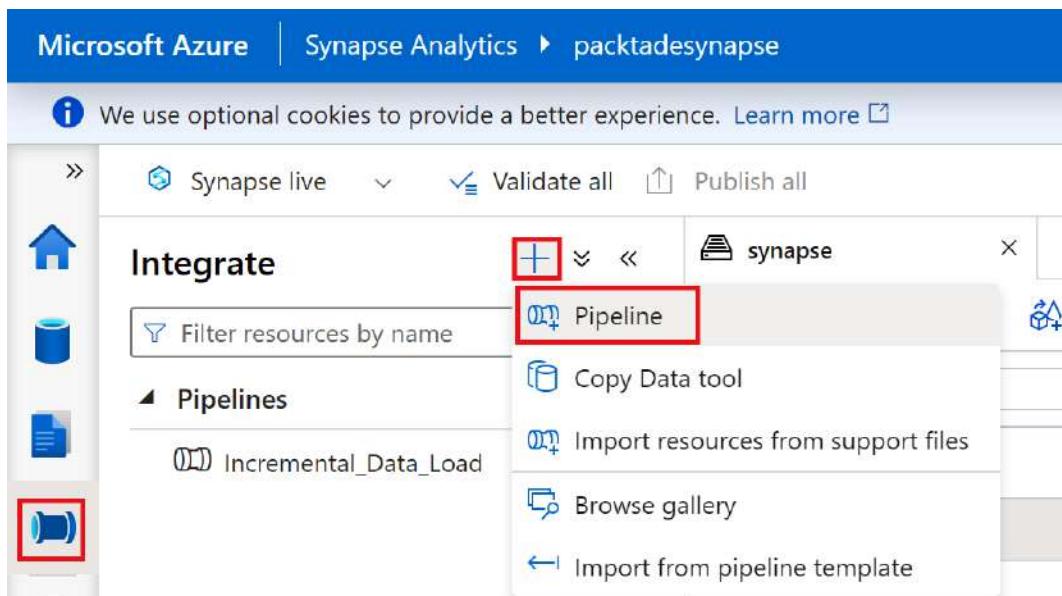


Figure 9.3 – Creating a pipeline

4. Name the pipeline **Copy_Data_Flow** by editing the **Name** property field on the right. Search for **Data flow** under **Activities**. Drag and drop the **Data flow** activity into the pipeline development area:

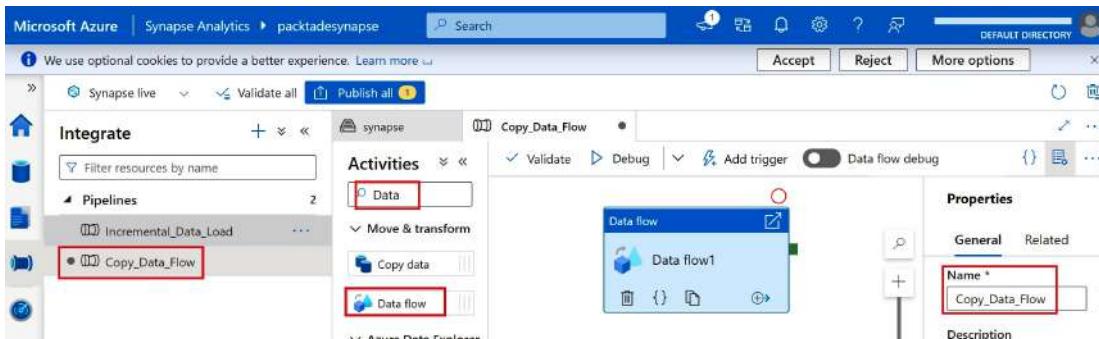


Figure 9.4 – Adding a data flow

5. Scroll down and under **Settings**, click on the **+ New** button to create a data flow:

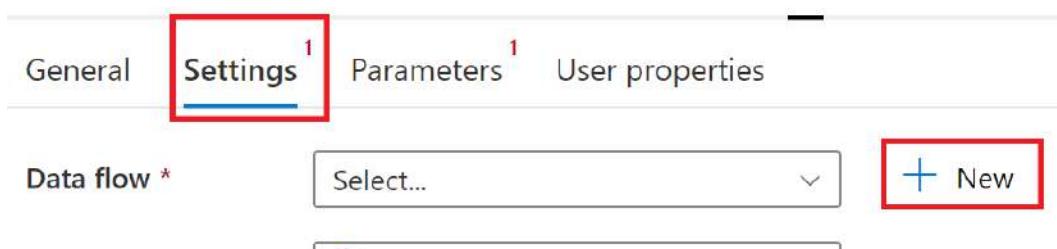


Figure 9.5 – Creating a data flow

6. Change the data flow name to **Copy_CSV_to_Parquet** on the left. Click on the **Add Source** button:



Figure 9.6 – Adding a source

7. Scroll down and name the output stream csv. Click on the + New button to create a new dataset:

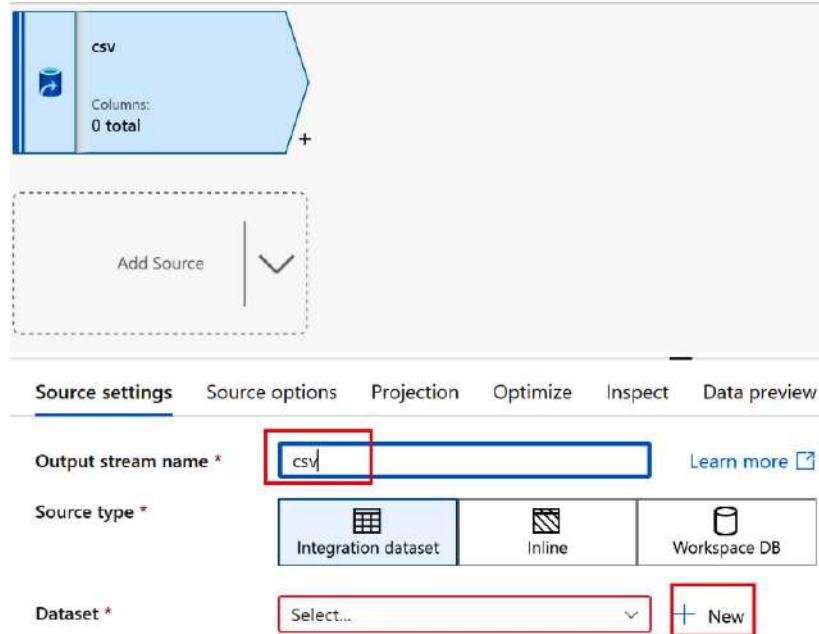


Figure 9.7 – Adding a dataset

8. Search for Azure Data Lake, select Azure Data Lake Storage Gen2, and click on Continue:

New integration dataset

In pipeline activities and data flows, reference a dataset to specify the location and structure of your data within a data store. [Learn more](#)

Select a data store

Azure Data Lake

All Azure Database File Generic protocol NoSQL Services and apps

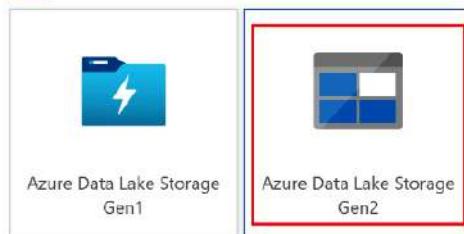


Figure 9.8 – Connecting to Azure Data Lake Gen 2

9. Select CSV as the data format, as the input file is CSV:



Figure 9.9 – Connecting to Azure Data Lake Gen 2

10. Provide the dataset name, `transactiontablet1`. Set **Linked service** to `packtadesynapse-WorkspaceDefaultStorage`. Set **File path** to `synapse/csv/transaction_table-t1.csv`, the location to which we uploaded the CSV file. Check the **First row as header** checkbox:

The screenshot shows the 'Set properties' dialog for a dataset named 'transactiontablet1'. The 'Name' field contains 'transactiontablet1'. The 'Linked service' dropdown is set to 'packtadesynapse-WorkspaceDefaultStorage'. Under 'Connect via integration runtime', 'AutoResolveIntegrationRuntime (Managed Virtual Network)' is selected. In the 'File path' section, the path 'synapse / CSV / transaction_table-t1.csv' is entered. The 'First row as header' checkbox is checked. The 'Import schema' section has 'From connection/store' selected. At the bottom, there are 'OK', 'Back', and 'Cancel' buttons, with 'OK' highlighted by a red box.

Figure 9.10 – Connecting to Azure Data Lake Gen 2

11. Click on the + button, search for **sink**, and select **Sink**. **Sink** is the destination component (transformation) that we will be copying the data to:

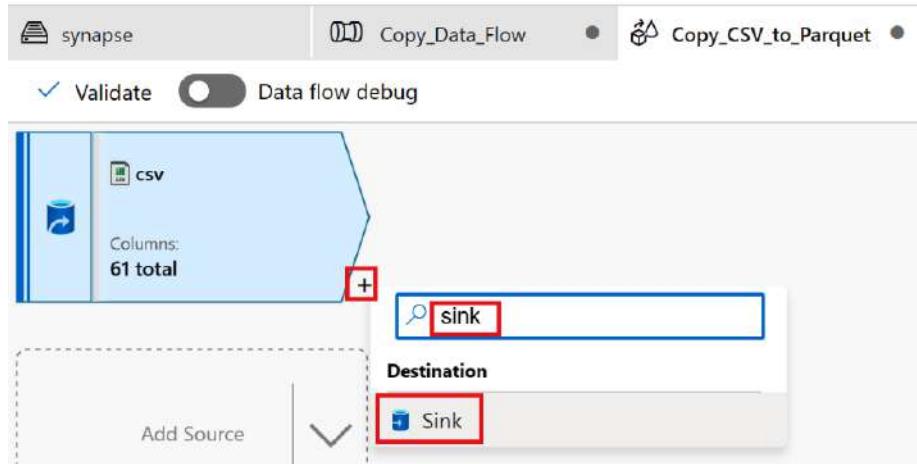


Figure 9.11 – Adding a sink

12. Set **Output stream name** to Parquet. Click + New:

The screenshot shows the Azure Synapse Dataflows interface with a 'csv' source connected to a 'Parquet' sink. The sink has 61 columns. Below the main canvas, there is a dashed box containing an 'Add Source' button. At the bottom, there is a detailed configuration pane for the sink:

- Sink** tab is selected.
- Output stream name ***: The input field contains 'Parquet', which is highlighted with a red box.
- Incoming stream ***: The dropdown menu shows 'CSV'.
- Sink type ***: The 'Integration dataset' option is selected.
- Dataset ***: The dropdown menu shows 'Select...', which is highlighted with a red box, and a '+ New' button to its right, also highlighted with a red box.
- Options**:
 - Allow schema drift ⓘ
 - Validate schema ⓘ

Figure 9.12 – Adding a sink

13. We need to define the destination dataset. Search for Azure Data Lake, select **Azure Data Lake Storage Gen2**, and click **Continue**:

New integration dataset

In pipeline activities and data flows, reference a dataset to specify the location and structure of your data within a data store. [Learn more](#)

Select a data store

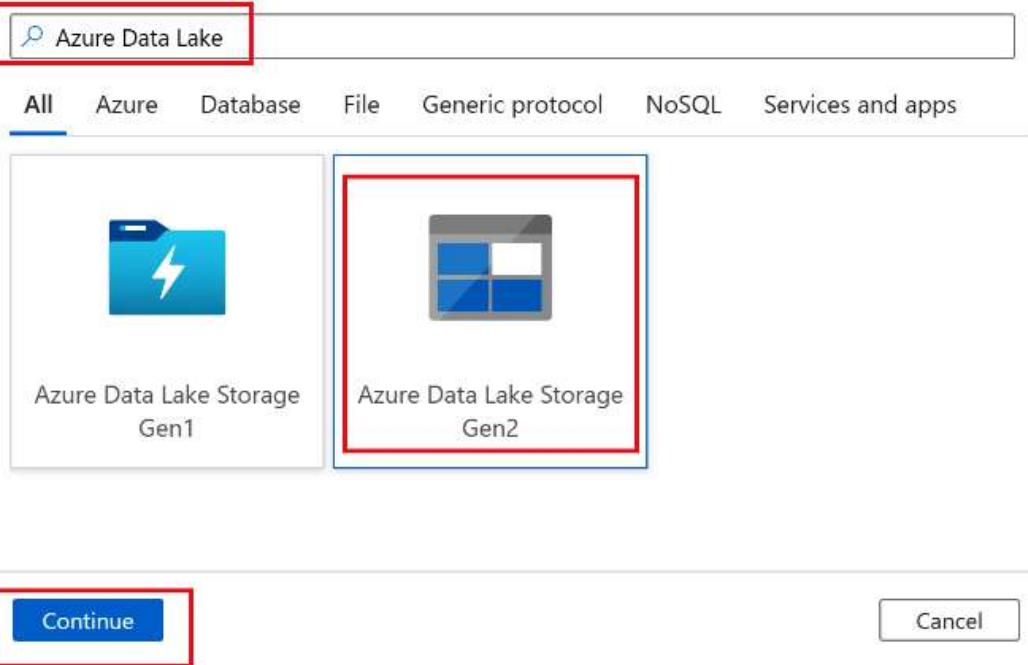


Figure 9.13 – Connecting to Azure Data Lake Gen 2

14. Set the format to **Parquet**:

Select format

Choose the format type of your data

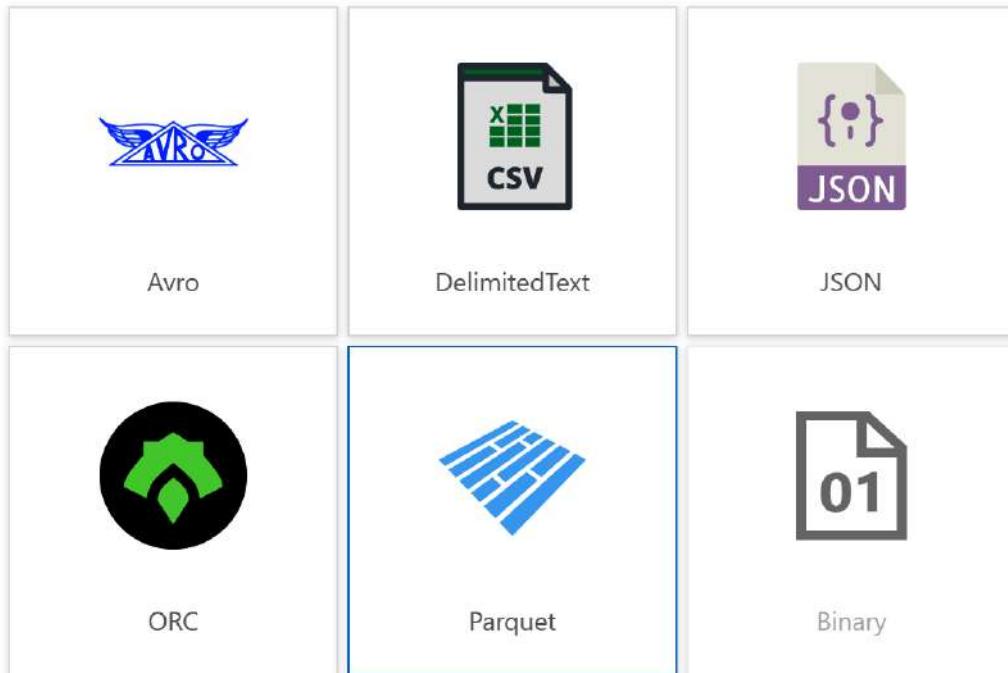


Figure 9.14 – Connecting to Azure Data Lake Gen 2

15. Provide the dataset name, `transactiontablet1parquet`. Set **Linked service** to `packtadesynapse-WorkspaceDefaultStorage`. Set **File path** to `synapse/transaction_table-t1-parquet/`, the location where we will store the Parquet file. Press **OK**.

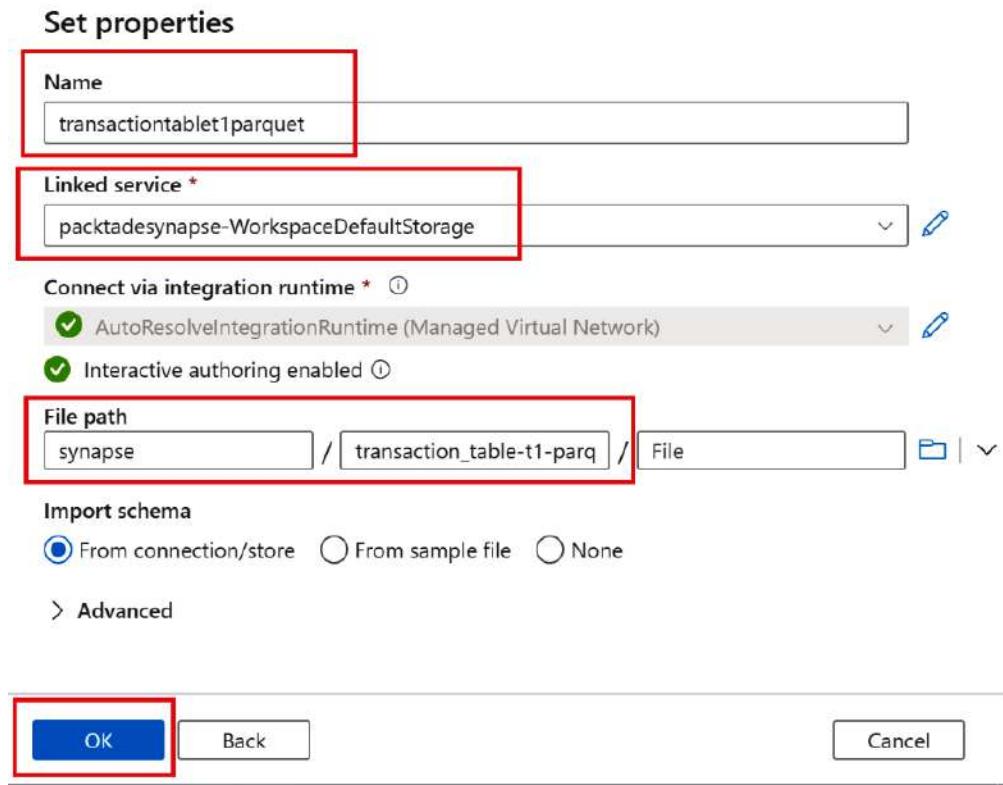


Figure 9.15 – Connecting to Azure Data Lake Gen 2

16. Hit the **Publish all** button. The data flow, datasets, and the pipeline will be published. Once they're published, go to the **Copy_Data_Flow** pipeline:

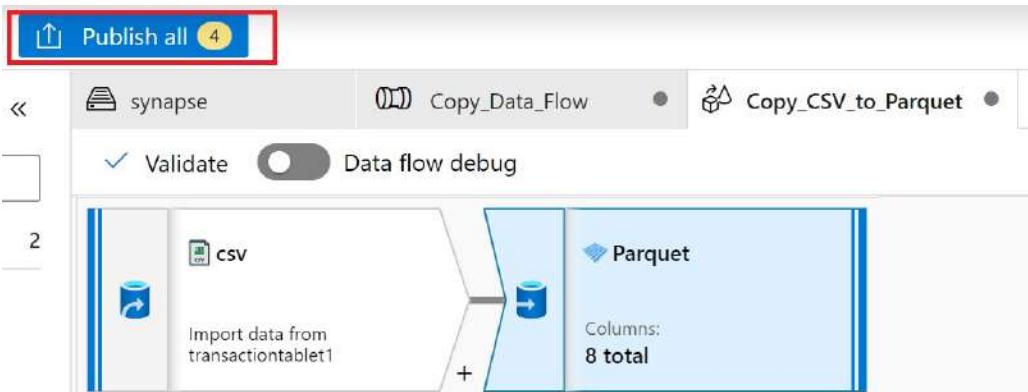


Figure 9.16 – Publishing the pipeline

17. Click on the **Add trigger** button and click **Trigger now** to trigger the execution of the pipeline:

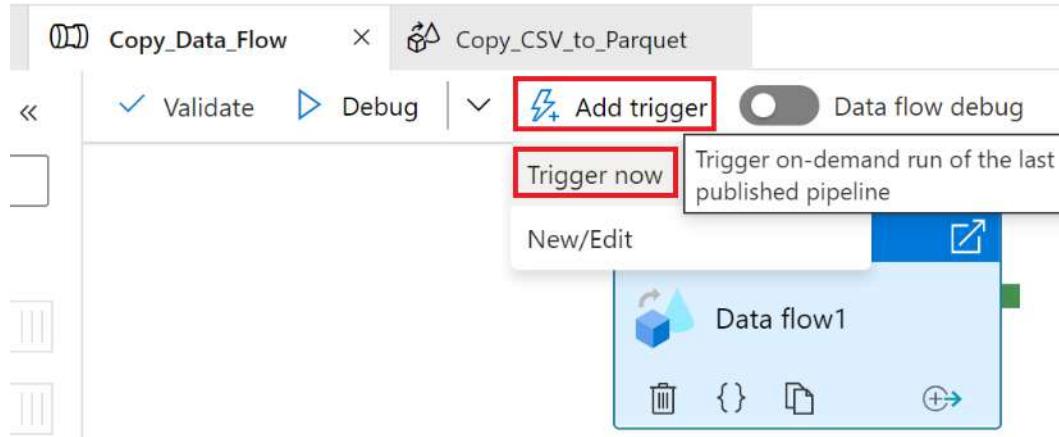


Figure 9.17 – The pipeline execution

18. Once the pipeline successfully runs, go to the storage section by clicking on the data icon (the blue cylinder) on the left-hand side and navigating to the **synapse (Primary)** folder. Right-click on the **transaction_table-t1-parquet** folder and click **Select TOP 100 rows**:

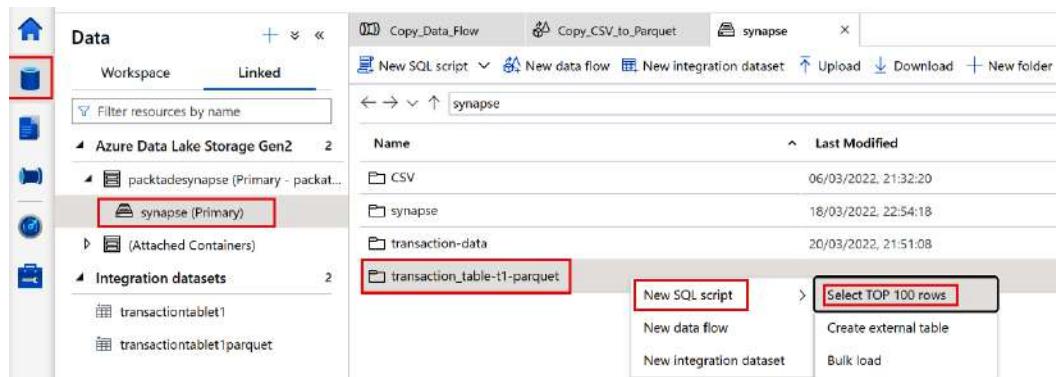


Figure 9.18 – Reading the transferred data

19. Set **File type** to **Parquet format** for querying:

Select TOP 100 rows

transaction_table-t1-parquet

Source folder format settings

Specify the format and layout of your data.

Folder path

https://packatadesynapse.dfs.core.windows.net/synapse/transaction_table-t1-parquet/



Figure 9.19 – Reading Parquet format

20. Hit the **Run** button in the query window. Data is successfully read via serverless SQL pool, confirming that the data transfer successfully completed:

A screenshot of a SQL query window titled 'Copy_Data_Flow'. The 'Run' button is highlighted with a red box. The query code is as follows:

```
1 -- This is auto-generated code
2 SELECT
3     TOP 100 *
4 FROM
5     OPENROWSET(
6         BULK 'https://packatadesynapse.dfs.core.windows.net/synapse/transaction_table-t1-parquet/**',
7         FORMAT = 'PARQUET'
8     ) AS [result]
```

The results pane shows a table with columns: tid, transaction_date, order_count, total_cost, sid, pid, c1, and c2. The data consists of 10 rows of transaction information.

tid	transaction_date	order_count	total_cost	sid	pid	c1	c2
170732	20211115	72	26091	4	7	70F825C6-A2B...	B02898
170733	20220312	54	13454	5	6	C4A5B069-C71...	BFE2C5
170734	20210628	60	15336	6	5	77C0E549-B824...	FEB695
170735	20220331	0	31866	7	4	691E6E33-8E5D...	A4BCB1
170736	20211125	51	5286	8	3	2B207737-5E1...	D87937

Figure 9.20 – Reading the transferred data

How it works...

The data flow does a simple conversion of the CSV file to a Parquet file. The data flow is linked to a Synapse integration pipeline called **Copy_Data_Flow**. The **Copy_Data_Flow** pipeline uses a data flow activity that calls the **Copy_CSV_to_Parquet** data flow, which does the data transfer. While this task can be done using a simple **Copy activity** as well (instead of data flow), the key advantage of a data flow is being able to perform transformations while moving the data from the source to the sink, which will be covered in more detail in the subsequent recipes in this chapter.

Performing data transformation using activities such as join, sort, and filter

A common scenario in data engineering pipelines is combining two or more files based on a column, filtering by column, sorting the results, and storing them for querying. We will perform the following actions to achieve this:

1. Read two CSV files
2. Use a **join transformation** to combine the two files based on a column
3. Use a **filter transformation** to filter the rows based on a condition
4. **Sort** the filtered data based on a column value and store the result in Parquet format

Getting ready

Create a Synapse Analytics workspace, as explained in the *Provisioning an Azure Synapse Analytics workspace* recipe of *Chapter 8, Processing Data Using Azure Synapse Analytics*:

1. Download the files - `transaction_table-t1.zip` and `transaction_table-t2.zip` - from <https://github.com/PacktPublishing/Azure-Data-Engineering-Cookbook-2nd-edition/upload/main/chapter9>.
2. Unzip them.
3. In the Synapse Analytics workspace, create a folder named `csv` (if it doesn't already exist) in the data lake account attached to the Synapse Analytics workspace. Upload the files, `transaction_table-t1.csv` and `transaction_table-t2.csv`, to the `csv` folder.

To see detailed example screenshots for a similar task, follow *steps 1 to 4* in the *How to do it...* section from the *Analyzing data using Serverless SQL pool* recipe of *Chapter 8, Processing Data Using Azure Synapse Analytics*.

How to do it...

In this recipe, we will perform the following:

- Create Azure Data Lake datasets connected to the `transaction_table-t1.csv` and `transaction_table-t2.csv` files
- Create a data flow activity in a pipeline
- Add **two source transformations** for reading CSV files
- Add a join transformation to combine the files based on a column named `tid`
- Add a filter transformation on the `transaction_date` column to filter the data by a particular date
- Add a sort transformation to sort the filtered data by the `total_cost` column
- Create a dataset with Parquet format as destination
- Add a sink and save the result in a dataset created using the Parquet format

The detailed steps to carry this out are as follows:

1. Log in to `portal.azure.com`, go to **All resources**, and search for **packtadesynapse**, the Synapse Analytics workspace created in the *Provisioning an Azure Synapse Analytics workspace* recipe of *Chapter 8, Processing Data Using Azure Synapse Analytics*. Click on the workspace. Click **Open Synapse Studio**:

The screenshot shows the Azure portal interface for a Synapse workspace named 'packtadesynapse'. The left sidebar contains navigation links for Overview, Activity log, Access control (IAM), Tags, Diagnose and solve problems, Settings (with options for Azure Active Directory, Properties, and Locks), Analytics pools (with options for SQL pools, Apache Spark pools, and Data Explorer pools (preview)), and Security (with an Encryption link). The main content area displays the 'Essentials' section with resource details:

Resource group (move)	:	PacktADESynapse
Status	:	Succeeded
Location	:	
Subscription (move)	:	Visual Studio Ultimate wit
Subscription ID	:	
Managed virtual network	:	Yes
Managed Identity object ...	:	
Workspace web URL	:	https://web.azuresynapse.
Tags (edit)	:	Click here to add tags

Below the essentials section is a 'Getting started' box with a red border, containing a blue hexagonal icon with a white 'S' symbol, the text 'Open Synapse Studio', and the subtext 'Start building your fully-integrated analytics solution and unlock new insights.' A blue 'Open' button with a right-pointing arrow is at the bottom of the box.

Figure 9.21 – Opening Synapse Studio

2. Click on the blue cylinder (the data symbol) on the left-hand side, which will take you to **Data** section. Click on the **Linked** tab. Expand the data lake account of the Synapse Analytics workspace (**packtadesynapse** for this example). Click on the **synapse (Primary)** container. Click on the **+ New folder** button and create a folder called `transaction_table-transformation-parquet`. We will use this folder as the destination to store the transformed data:

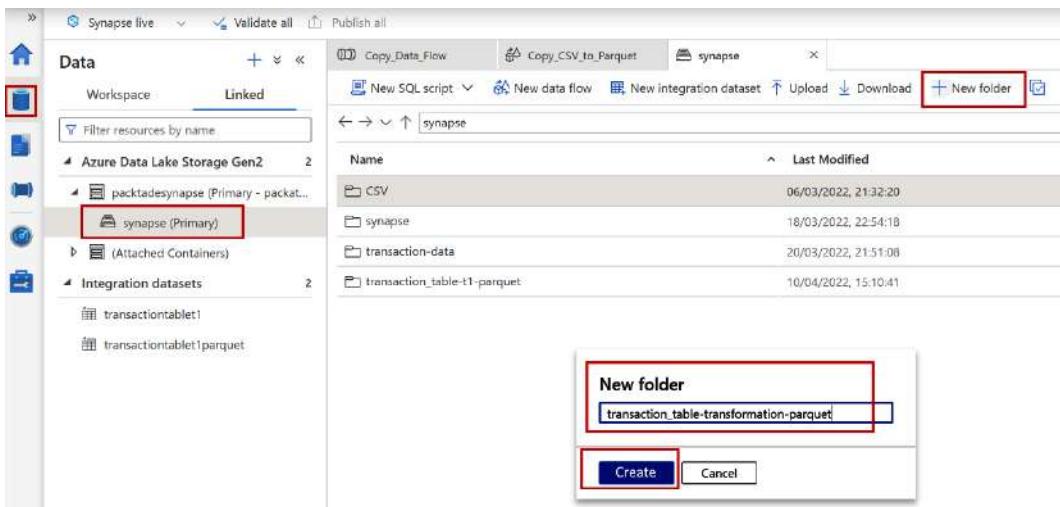


Figure 9.22 – Creating the Synapse Studio folder

3. Create a pipeline named DataFlow-Transformation. Add a data flow activity in the pipeline. Create a new data flow named SortFilterDataFlow by clicking on the + New button under the data flow activity settings. To see detailed example screenshots for a similar task, follow steps 3 to 5 in the *How to do it...* section from the *Copying data using a Synapse data flow* recipe:

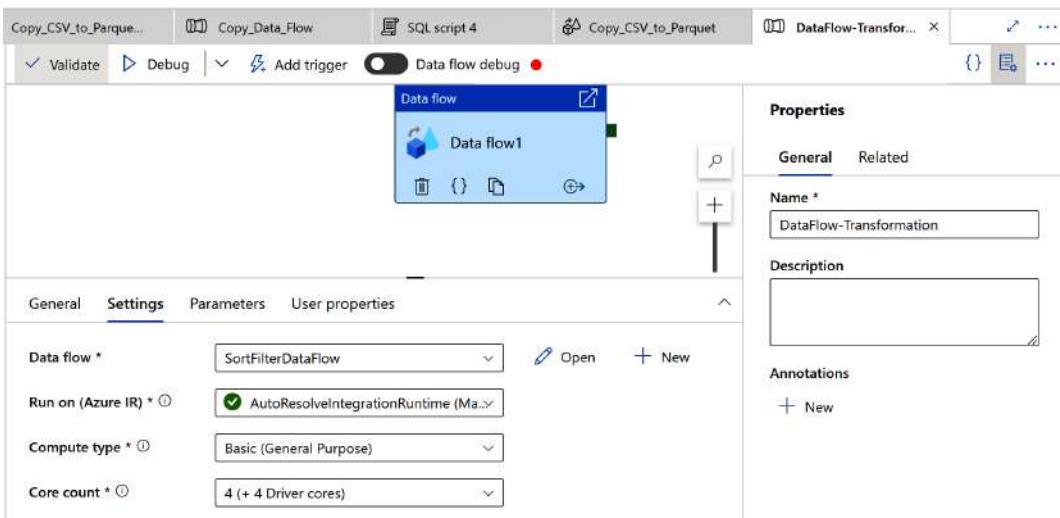


Figure 9.23 – Creating the Dataflow-Transformation pipeline

4. Add a source transformation named `transactiontable1`, click on the **+ New** button, and create a dataset named `transactiontable1`, as explained in steps 6 to 10 in the *How to do it...* section from the *Copying data using a Synapse data flow* recipe:

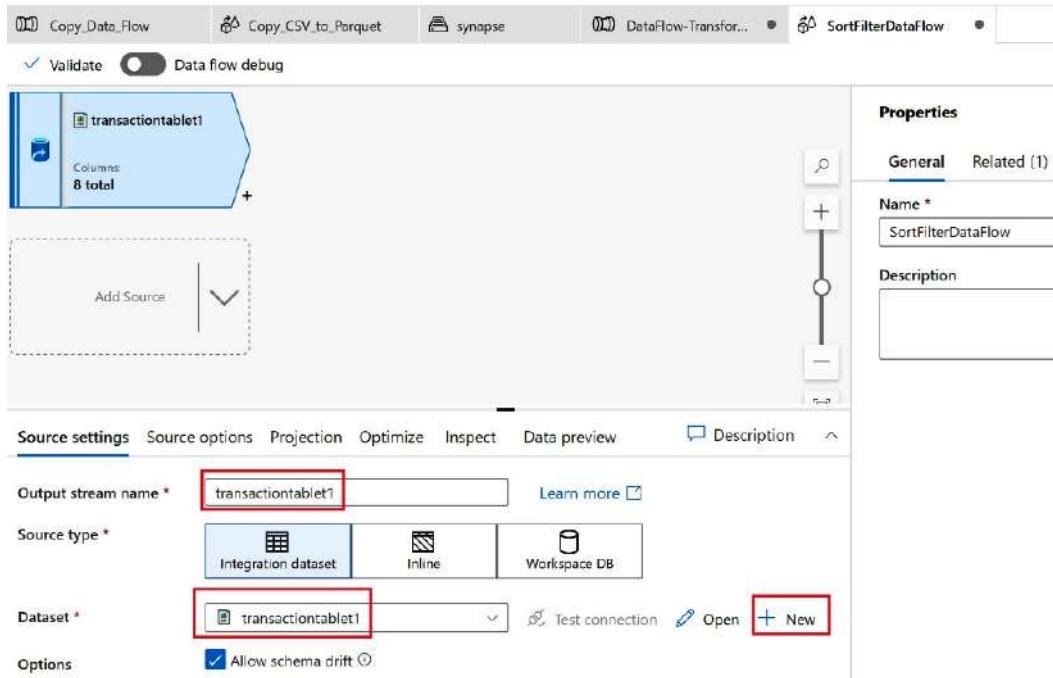


Figure 9.24 – Creating the Dataflow-Transformation pipeline

5. Click on the **Add Source** button.

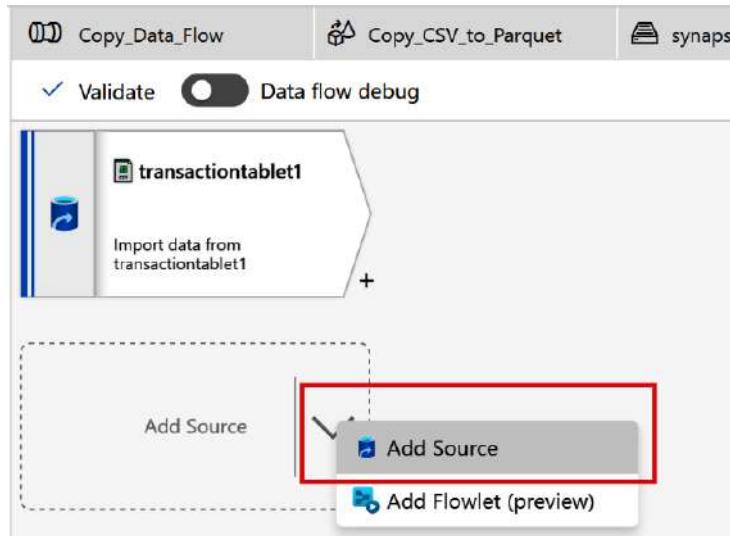


Figure 9.25 – Adding a source

6. Name the source transformation `transactiontable2`. Click on the `+ New` button to create a new dataset to link to `synapse/csv/ transaction_table-t2.csv`:

The screenshot shows the 'Source settings' blade for adding a new source dataset. At the top, there is a preview of the data flow with a 'transactiontable1' dataset and a 'transactiontable2' dataset (which is currently empty, showing '0 total' columns). Below this, a navigation bar includes 'Source settings' (selected), 'Source options', 'Projection', 'Optimize', 'Inspect', and 'Data preview'. The 'Source settings' tab is active. Under 'Output stream name *', the value 'transactiontable2' is entered in a text input field, which is highlighted with a red rectangle. Next to it is a 'Learn more' link. Under 'Source type *', there are three options: 'Integration dataset' (selected, highlighted with a blue background), 'Inline', and 'Workspace DB'. Under 'Dataset *', a dropdown menu is shown with the value 'Select...' and a '+' New button, both of which are highlighted with red rectangles. At the bottom, there is an 'Options' section with a checked checkbox for 'Allow schema drift'.

Figure 9.26 – Adding the source

7. Create a dataset named `transactiontablet2` linking to Azure Data Lake Storage Gen2 account attached to the Synapse Analytics workspace, with the location set to `synapse/csv/transaction_table-t2.csv` and the format set to `csv`. Refer to steps 7 to 10 in the *How to do it...* section from the *Copying data using a Synapse data flow* recipe:

Set properties

Name
transactiontablet2

Linked service *
packtadesynapse-WorkspaceDefaultStorage

Connect via integration runtime * ⓘ
 AutoResolveIntegrationRuntime (Managed Virtual Network)
 Interactive authoring enabled ⓘ

File path
synapse / csv / transaction_table-t2.csv

First row as header

Import schema
 From connection/store From sample file None

> Advanced

OK Back Cancel

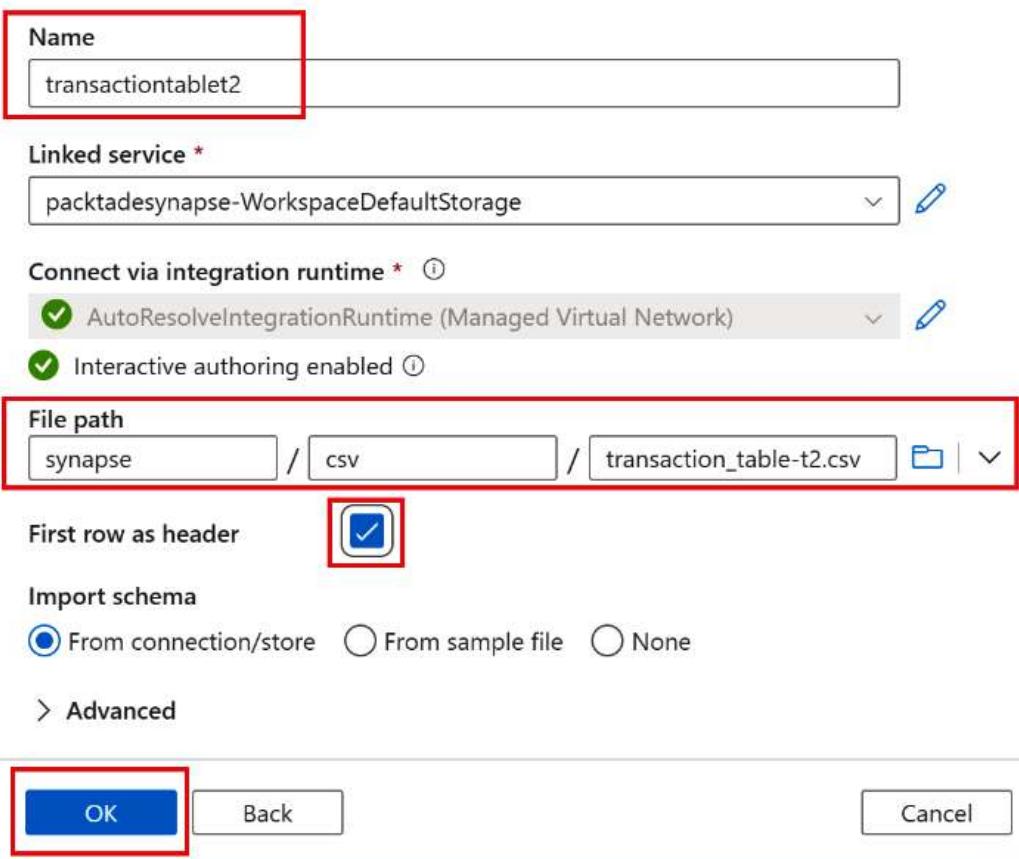


Figure 9.27 – Configuring the data source

8. Click on the + button on the `transactiontablet1` source and select the **Join** transformation:

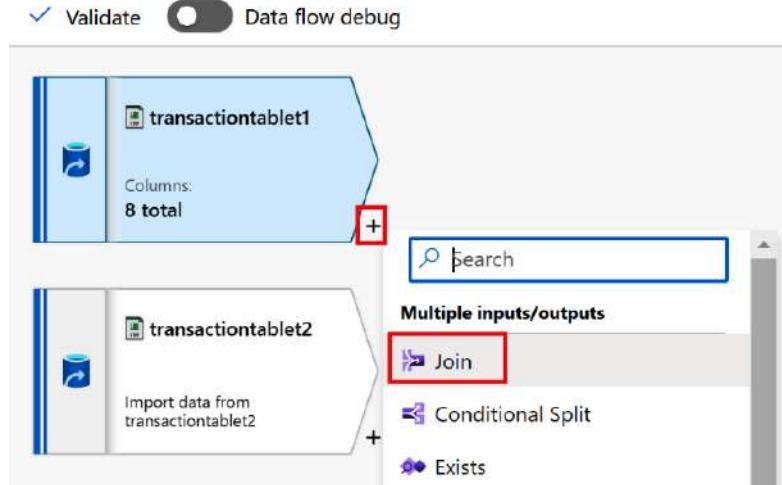


Figure 9.28 – Adding join

- Set the **Left stream** dataset to **transactiontable1** and the **Right stream** dataset to **transactiontable2**. Select **Inner** as the join type and join using the **tid** column, which exists on both the datasets.

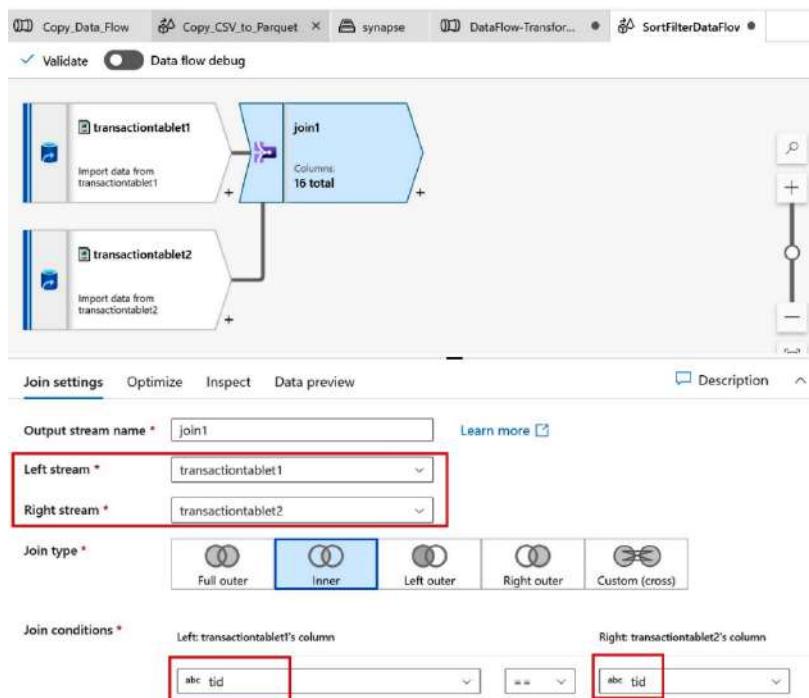


Figure 9.29 – Configuring the join transformation

10. Click on the + button on the right-hand side of the **join1** transformation and select the **Filter** transformation:

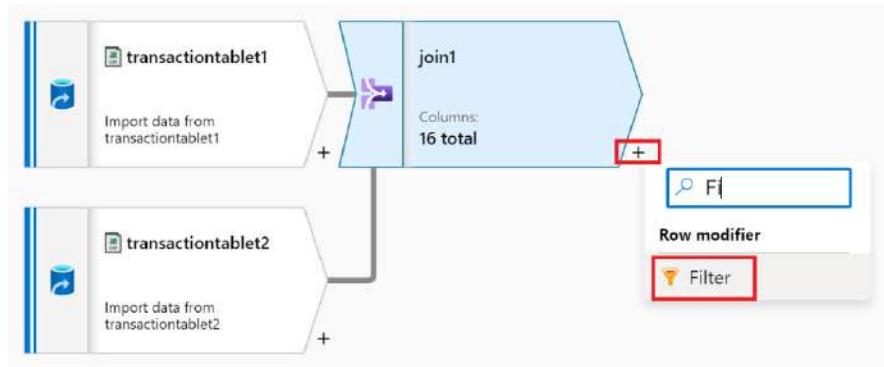


Figure 9.30 – Adding the filter transformation

11. Type `transactiontable1@transaction_date=="20210915"` into the **Filter on** text box.

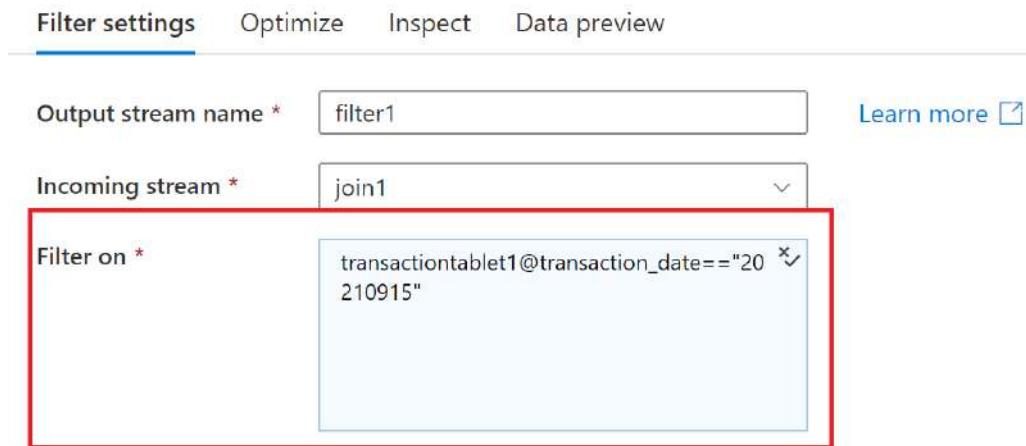


Figure 9.31 – Configuring the filter transformation

12. Click on the + button on the right-hand side of the **filter1** transformation and select the **Sort** transformation:

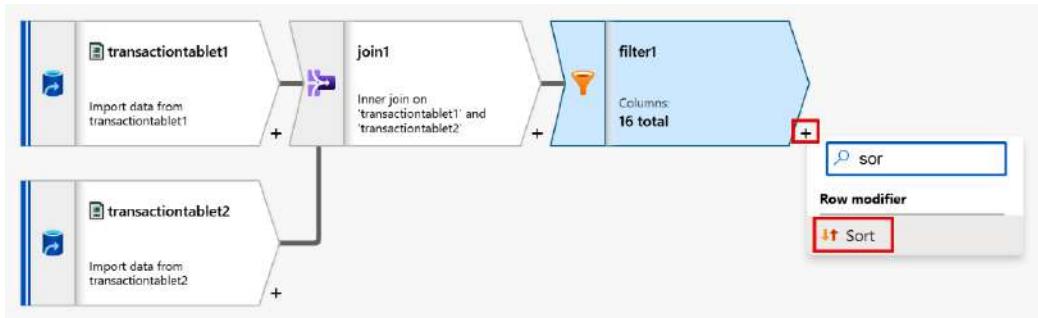


Figure 9.32 – Add sort transformation

13. Under **Sort conditions**, select **transactiontable1@total_cost** for the **filter1's column** dropdown and set **Order** to **Descending**:

Sort conditions *		filter1's column	Order	Nulls first
abc	transactiontable1@total_cost	Descendi...	<input checked="" type="checkbox"/>	+ Delete

Figure 9.33 – Adding the filter conditions

14. Hit the **+** button on the right-hand side of the **sort1** transformation and add a sink. Under the **Sink** properties, click on the **+ New** button and add a new dataset named **transformationparquet** linking to the Azure Data Lake Gen2 account attached to the Synapse Analytics workspace with the location set to **synapse/transaction_table-transformation-parquet** folder and the format set to Parquet. Refer to steps 11 to 15 in the *How to do it...* section from *Copying data using a Synapse data flow* recipe to see detailed example screenshots for a similar task. Once complete, the sink should appear as shown in the following screenshot:

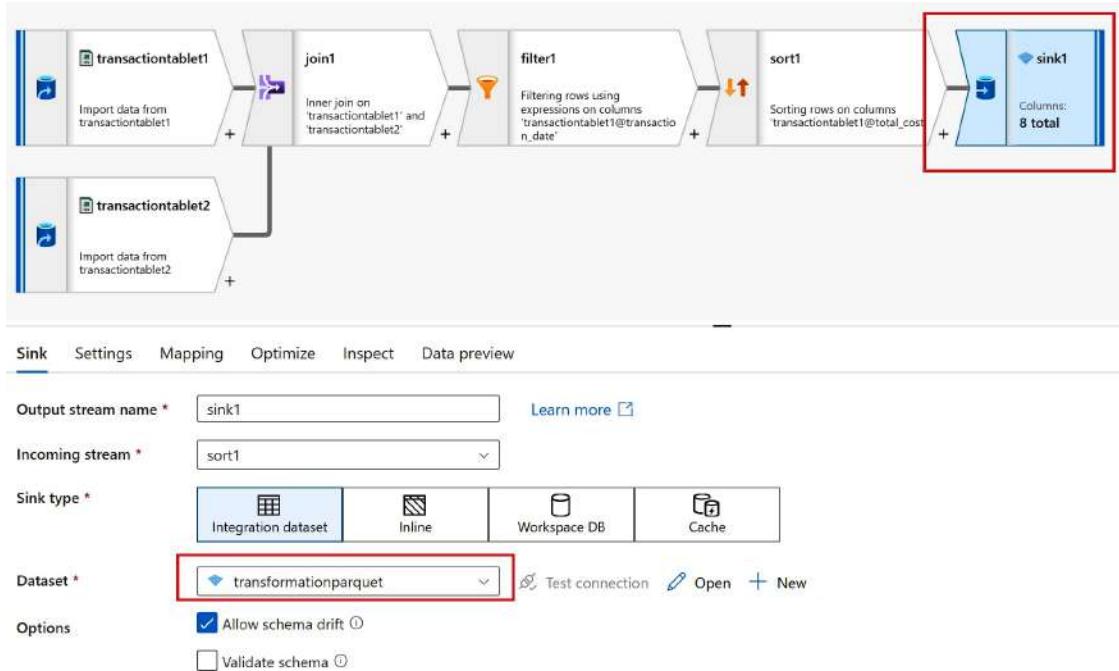


Figure 9.34 – Adding the sink

15. Switch to the **DataFlow-Transformation** pipeline and hit the **Publish all** button. Once published, hit the **Trigger now** button under **Add trigger**. After successful execution, go to the **synapse/transaction_table-transformation-parquet** folder in the Synapse Analytics workspace, right-click on it, and select the top 100 rows. Refer to steps 16 to 20 in the *How to do it...* section from *Copying data using a Synapse data flow* recipe to see detailed example screenshots for a similar task. The result is shown in the following screenshot:

The screenshot shows the Azure Synapse Analytics Studio interface. At the top, there's a toolbar with 'Run' (highlighted with a red box), 'Undo', 'Publish', 'Query plan', 'Connect to' (set to 'Built-in'), 'Use database' (set to 'master'), and a refresh button. Below the toolbar is a code editor window containing the following T-SQL script:

```

1 -- This is auto-generated code
2 SELECT
3     TOP 100 *
4 FROM
5     OPENROWSET(
6         BULK 'https://packatadesynapse.dfs.core.windows.net/synapse/transaction_table-transformation-parquet/**'
7         FORMAT = 'PARQUET'
8     ) AS [result]
9

```

Below the code editor is a results grid. The 'Results' tab is selected. The results show a table with the following data:

tid	transaction_date	order_count	total_cost	sid	pid	c1	c2
271109	20210915	57	9729	1	10	E7B2FD1C-2E8...	5E9B4A
191913	20210915	71	9686	1	10	CD9D4E1C-0A...	1F2214
163707	20210915	20	9684	1	9	A6F14644-B6A...	33802F
265287	20210915	70	9608	2	9	A8D75C06-477...	40544D

The 'transaction_date' column for all rows is highlighted with a red box.

Figure 9.35 – The result

How it works...

A join transformation helped us to combine the `transaction_table-t1.csv` and `transaction_table-t2.csv` files based on the `tid` column. The combined data from both files was filtered using the filter transformation and the `transaction_table-t1.csv` file's `transaction_date` column was filtered for the date `20210915`. The sort transformation sorted the filtered rows based on the `transaction_table-t1.csv` file's `total_cost` column. A sink transformation was linked to a dataset in Parquet format, which meant that the sorted and filtered data from the CSV file was seamlessly converted and stored in Parquet format too.

Data flows have plenty of transformation options available. You can explore more examples at <https://docs.microsoft.com/en-us/azure/data-factory/data-flow-transformation-overview>.

Monitoring data flows and pipelines

Azure Synapse Analytics provides a user-friendly interface out of the box for monitoring the pipeline and data flow runs. In this recipe, we will track a data flow execution and understand the transformation execution timings.

Getting ready

Create a Synapse Analytics workspace, as explained in the *Provisioning an Azure Synapse Analytics workspace* recipe of *Chapter 8, Processing Data Using Azure Synapse Analytics*.

Complete the *Performing data transformation using activities such as join, sort, and filter* recipe in this chapter to create some data flow runs.

How to do it...

Perform the following steps to monitor data flows:

1. Log in to `portal.azure.com`, go to **All resources**, and search for **packtadesynapse**. Click on the workspace. Click on **Open Synapse Studio**. Click on the monitor button (a speedometer-like circular button) on the left-hand side. By default, it will give you the information about pipeline runs from the last 24 hours:

The screenshot shows the Microsoft Azure Synapse Analytics Pipeline runs page. The URL is `Microsoft Azure | Synapse Analytics > packtadesynapse`. The left sidebar shows categories like Analytics pools, Activities, and Integration. Under Integration, the 'Pipeline runs' option is selected and highlighted with a red box. The main area displays a table of pipeline runs with the following data:

Pipeline name	Run start	Run end	Duration	Triggered by	Status	Error	Run	Parameters
DataFlow-Transformation	4/10/22, 6:57:00 PM	4/10/22, 7:01:25 PM	00:04:25	Manual trigger	Succeeded		Original	
Incremental_Data_Load	4/10/22, 5:00:00 PM	4/10/22, 5:03:01 PM	00:03:00	Incremental_Data_Load	Failed		Original	
Copy_Data_Flow	4/10/22, 12:33:33 PM	4/10/22, 12:37:12 PM	00:03:39	Manual trigger	Succeeded		Original	

Figure 9.36 – The pipeline runs

2. Click on the **DataFlow-Transformation** pipeline with the most recent run date (for me, it's **4/10/22**).

The screenshot shows the Microsoft Azure Synapse Analytics Pipeline execution data page for the 'DataFlow-Transformation' pipeline. The URL is similar to Figure 9.36. The table shows the following data:

Pipeline name	Run start	Run end	Duration	Triggered by	Status
DataFlow-Transformation	4/10/22, 6:57:00 PM	4/10/22, 7:01:25 PM	00:04:25	Manual trigger	Succeeded
Copy_Data_Flow	4/10/22, 12:33:33 PM	4/10/22, 12:37:12 PM	00:03:39	Manual trigger	Succeeded

Figure 9.37 – Opening the pipeline execution data

3. Under **Activity runs**, hover your mouse over the **Data flow1** activity and click on the glasses icon:

The screenshot shows the 'Activity runs' section of the DataFlow-Transformation pipeline. A tooltip for 'Data flow1' is displayed, showing its status as 'Succeeded'. The 'Activity runs' table has one item:

Activity name	Activity type	Run start	Duration	Status	Error	Integration runtime
Data flow1	Data flow	4/10/22, 6:57:02 PM	00:04:23	Succeeded		AutoResolveIntegrationRuntime (East US)

Figure 9.38 – Opening the data flow execution details

4. All the transformations under our activity are displayed. Click on the **Stages** icon to identify the transformation that took the most time:

The screenshot shows the 'Data flow1' execution details. It displays a sequence of stages: a source (Azure Blob Storage), followed by five processing stages (represented by icons: purple arrow, orange funnel, red double-headed arrow, blue square), and a sink (Azure Blob Storage). Below the stages, a lineage diagram shows the flow of data from the source through the stages to the sink. The 'Stages' icon is highlighted with a red box.

Figure 9.39 – The data flow execution details

5. The total execution time was 25 seconds, of which sink and sort operations took 20 seconds. We also noticed that there were **1009** rows sorted and written to sink. Click **Close**:

Stages

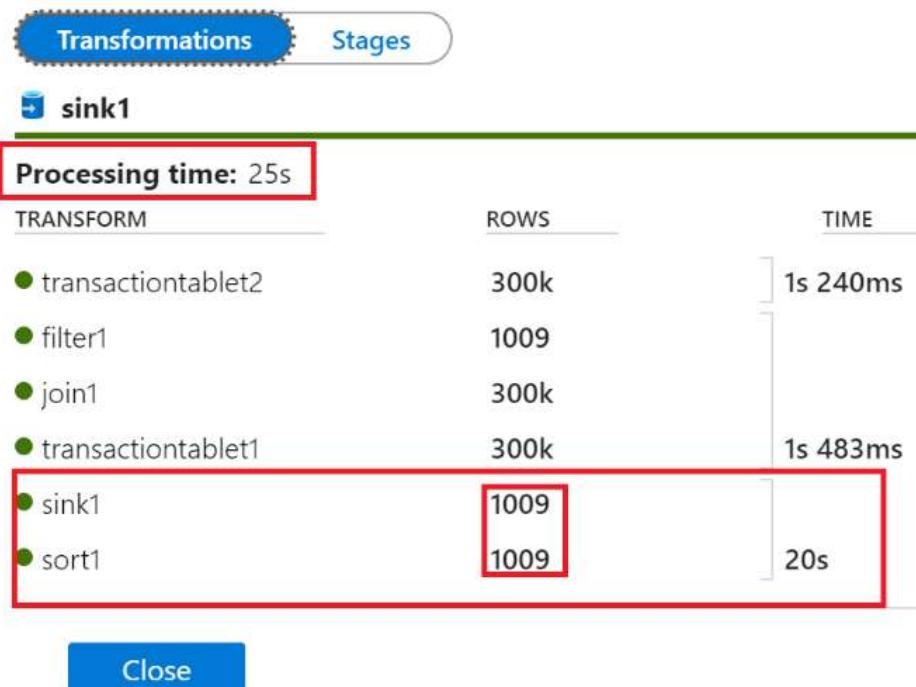


Figure 9.40 – The transformation-level execution details

How it works...

The Azure Synapse Analytics out-of-the-box monitoring solution records details about all pipeline execution runs and details about the activities inside the pipeline too. In this recipe, we saw that we could quickly identify even a slow transformation inside a data flow activity in a matter of a few clicks. The monitoring data is by default stored for 45 days and can be retained for a longer duration by integrating Synapse Analytics diagnostics data with Azure Log Analytics or an Azure Data Lake Storage account.

Configuring partitions to optimize data flows

Data flows, by default, create partitions behind the scenes to make transformation activities such as join, filter, and sort run faster. Partitions split the large data into multiple smaller pieces so that the backend processes in the data flows can divide and conquer their tasks and finish the execution quickly.

In this recipe, we will take a slow-running data flow and adjust the partitioning to reduce the execution time.

Getting ready

Create a Synapse Analytics workspace as explained in the *Provisioning an Azure Synapse Analytics workspace* recipe.

Complete the *Performing data transformation using activities such as join, sort, and filter* and *Monitoring data flows and pipelines* recipes in this chapter.

How to do it...

Perform the following steps to optimize data flows:

1. Log in to `portal.azure.com`, go to **All resources**, and search for **packtadesynapse**. Click on the workspace. Click **Open Synapse Studio**. Click on the monitor button (the speedometer-like circular button) on the left-hand side. Refer to *steps 2 to 5 of the How to do it... section from the Monitoring data flows and pipelines recipe* and perform the following actions:
 - I. Click on the latest run of the **DataFlow-Transformation** pipeline.
 - II. Click on the **Data flow1** activity details.
 - III. Click on the **Stages** icon to get the transformation level breakdown of the execution times. Notice that the sort and sink operations consume 80% of the execution time, taking 20 seconds.
2. Click on the **Sort1** transformation in the monitoring window. Notice the following:
 - The total number of rows processed by the transformation is only 1,009
 - There are about 200 partitions
 - Each partition has around 4 to 6 rows

Let's make some changes to partitioning in the sort transformation. Click on the **Edit transformation** button at the top of the page:

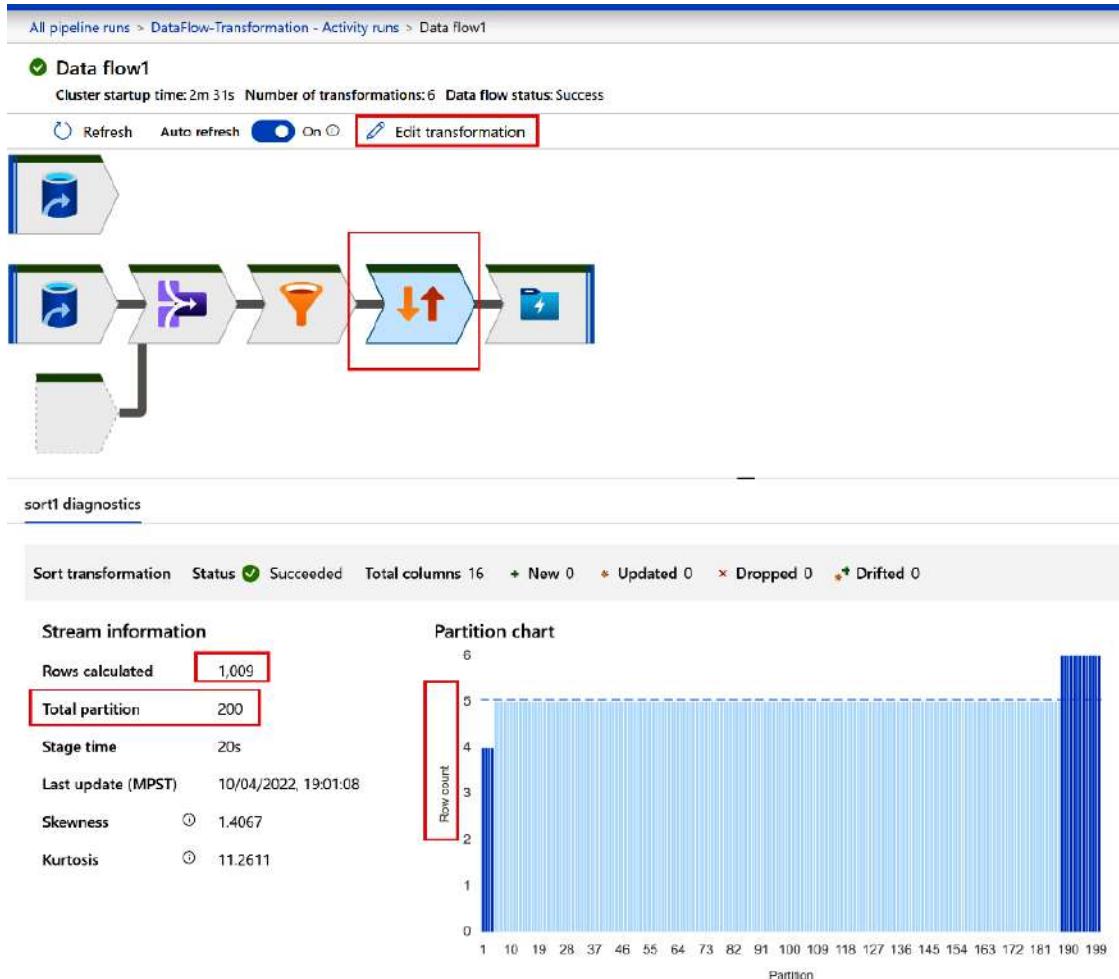


Figure 9.41 – The partition statistics

3. Click on the **sort1** transformation. Go to the **Optimize** section. Select **Single partition** (instead of the existing setting, **Use current partitioning**). Hit the **Publish all** button:

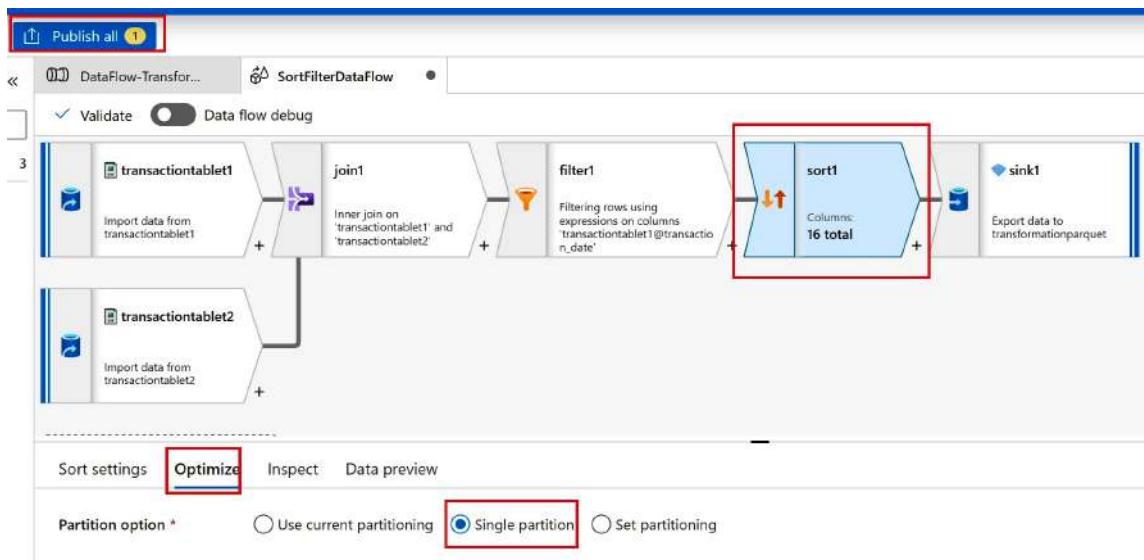


Figure 9.42 – Setting a single partition

4. Go to **DataFlow-Transformation**, click on **Add trigger**, and click **Trigger now**:

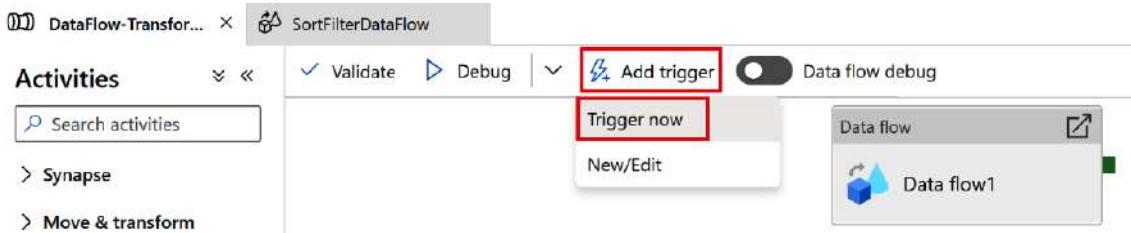


Figure 9.43 – The Trigger now function

5. Check the notification section (the bell icon) on the right-hand corner of the screen. The run completion will be indicated via a notification message. Click **View pipeline run**:

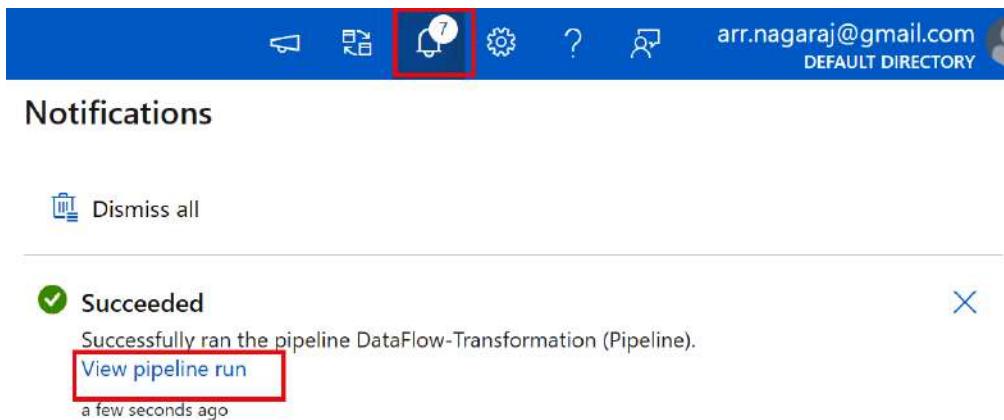


Figure 9.44 – Viewing the pipeline run

- Click on the **Dataflow1** activity details. Click on the **Stages** icon to get the transformation level breakdown of the execution times, as we did previously:
 - Notice that the total execution time has been reduced to 8 seconds
 - The sort and sink executions were reduced to 2.8 seconds, compared to the earlier duration of 20 seconds

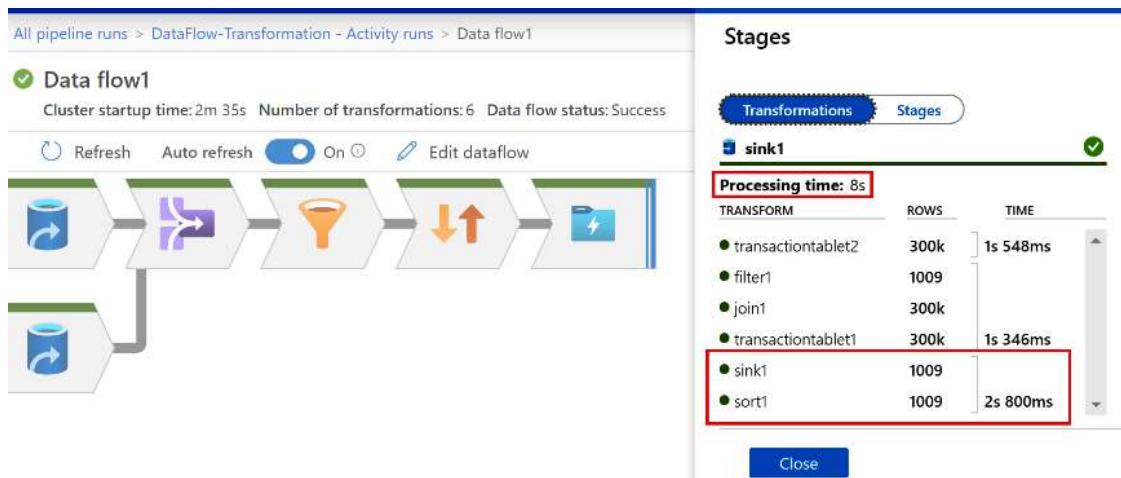


Figure 9.45 – The execution timing of the pipeline run

How it works...

Behind the scenes, Synapse Analytics data flows use tasks in Spark clusters to perform data flow transformations. When observing the partition statistics on the **sink1** transformation in *step 2* of the *How to do it...* section, we noticed that 1,009 rows were split across 200 partitions. Typically, we would like to have at least a few thousand rows per partition (or 10 MB to 100 MB in size). Having 4 to 6 rows per partition makes any transformation slow and hence, the sort operation was slow as well. Having too many partitions implies that the backend jobs spend a lot of time creating many partitions. This becomes overkill when there are just a few rows per partition and most of the time is spent on creating partitions rather than processing the data inside them.

Switching the partition settings to **Single partition** via the sink transformation's **Optimize** setting creates a single partition for all 1,009 rows, performs the sort activity in a single partition, and returns the results quickly. Had there been, say, a few hundred thousand or a few million rows for the sort activity, having multiple partitions would have been a better bet.

Data flows for each transformation, by default, use the setting called **current partitioning**, which implies that the data flow will estimate the optimal number of partitions for that transformation based on the size of the data. Setting a single partition as done in this recipe is *not* recommended for all scenarios – however, it is equally important to track the partition count for transformations and make adjustments if required.

Parameterizing Synapse data flows

Adding parameters to data flows provides flexibility and agility for data flows while performing their transformations. In this recipe, we will create a data flow that accepts a parameter, filters the data based on the value passed in the parameter, and copies the data to Parquet files.

Getting ready

Create a Synapse Analytics workspace as explained in the *Provisioning an Azure Synapse Analytics workspace* recipe in *Chapter 8, Processing Data Using Azure Synapse Analytics*.

Complete the *Copying data using a Synapse data flow* recipe in this chapter to create the **Copy_CSV_to_Parquet** data flow..

How to do it...

In this recipe, we will add a parameter to the **Copy_CSV_to_Parquet** dataflow (which was created in the *Copying data using a Synapse data flow* recipe), and make the **Copy_CSV_to_Parquet** data flow copy selective rows based on the value passed to the parameter by the **Copy_Data_Flow** pipeline (created in the *Copying data using a Synapse data flow* recipe). The **Copy_CSV_to_Parquet** data flow currently copies all 300,000 rows present in the `transaction_tablet1.csv` file and adds the filter based on parameter that will result in lesser number of rows being copied:

1. Log in to `portal.azure.com`, go to **All resources**, and search for **packtadesynapse**. Click on the workspace. Click **Open Synapse Studio**. Click on the **Develop** icon (the notebook-like symbol) on the left-hand side of the screen. Expand **Data flows** and click **Copy_CSV_to_Parquet**. Under **Parameters**, click **+ New** and add a new parameter with the name `sid`. Set **Default value** to `0`:

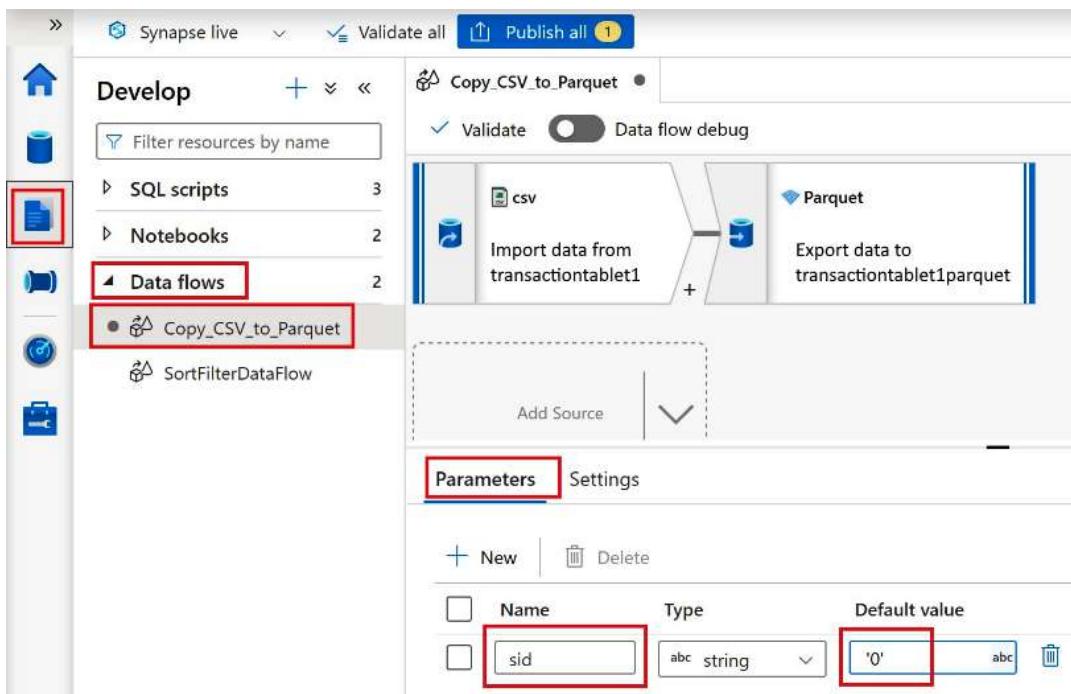


Figure 9.46 – Adding a parameter

2. Click on the + button on the right-hand side of the source transformation named **csv**, search for a filter transformation, and add a filter transformation:

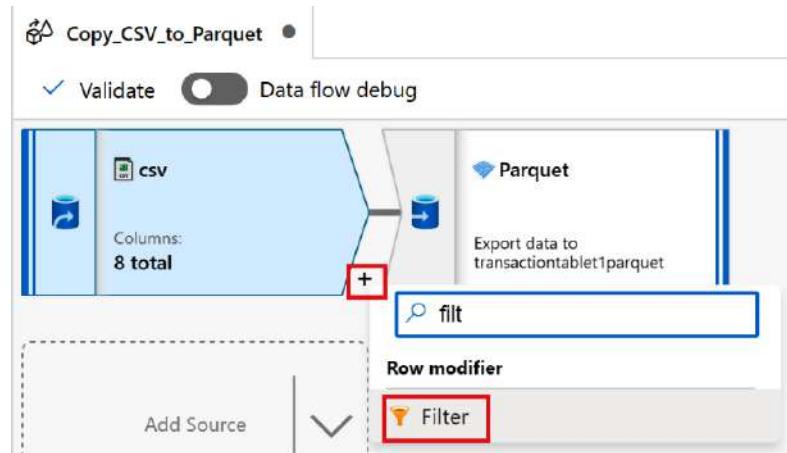


Figure 9.47 – Adding the filter

3. Click on the **Filter on** textbox under **Filter settings**. Click on the **Open expression builder** link found below the textbox:

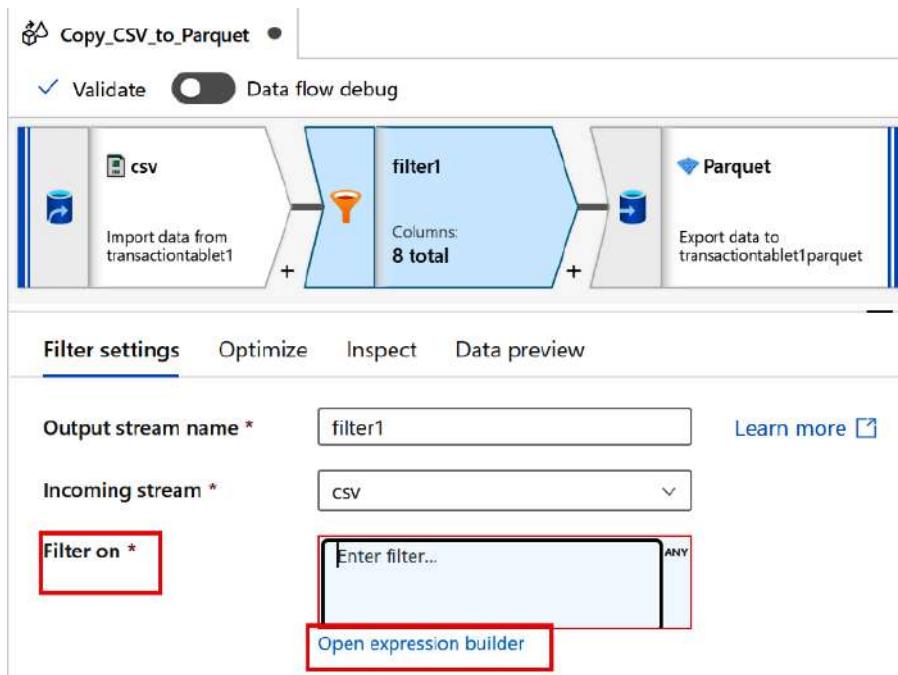


Figure 9.48 – Adding dynamic content

4. In the expression text box, type the column name that we will be filtering the parameter against; in our case, it is `sid`. Click on `==` (double equal to operator). Click **Parameters** on the bottom-left-hand side and select the `sid` parameter. The expression textbox should read as shown in the following screenshot. Click **Save and finish**:

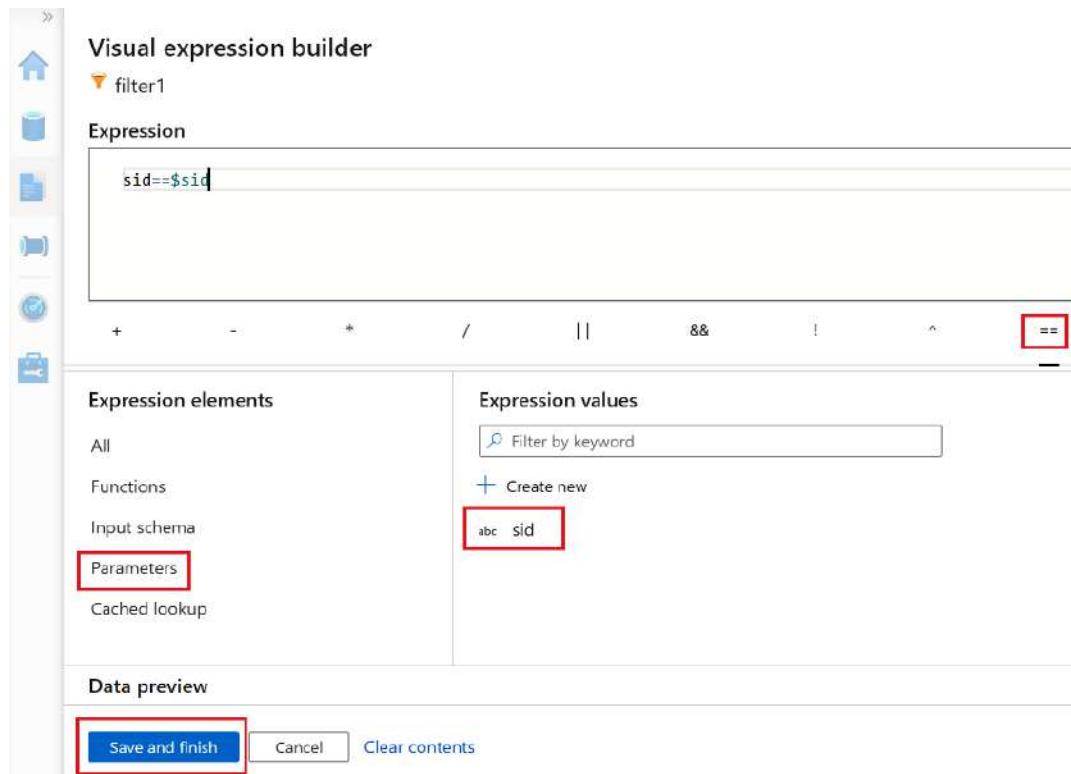


Figure 9.49 – Adding dynamic content

5. Click on the **Integrate** icon (the pipe-like symbol) on the left-hand side of Synapse Studio. Click on the **Copy_Data_Flow** pipeline. Click on the **Data flow1** task and go to the **Parameters** section of the task. The filename parameter we added to the data flow in *step 1* appears here. Click on the **Value** box, select **Pipeline expression**, and check the **Expression** checkbox:

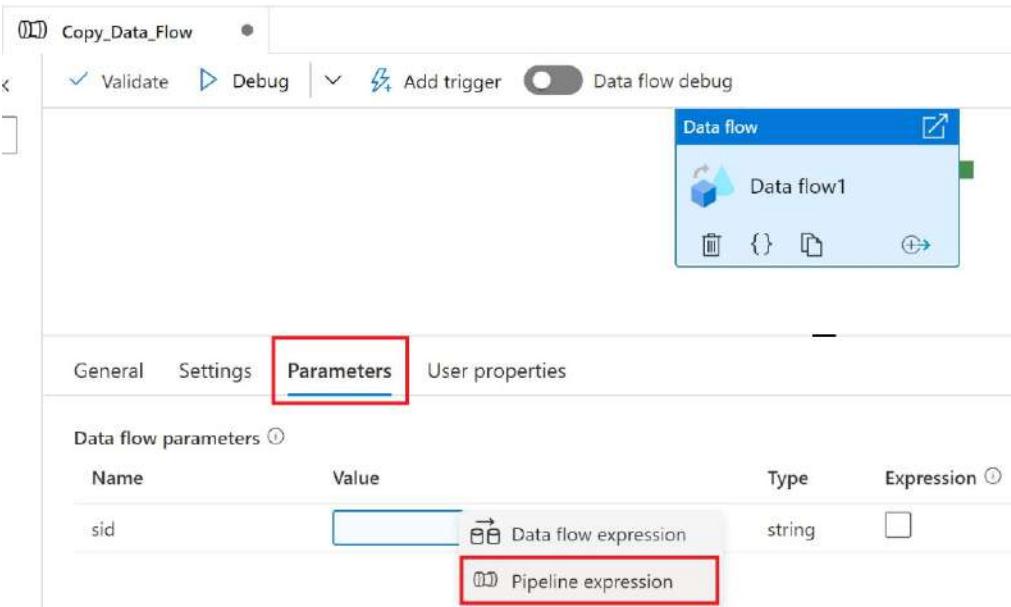


Figure 9.50 – Adding the pipeline expression

6. Type in '10' under dynamic content and click OK:

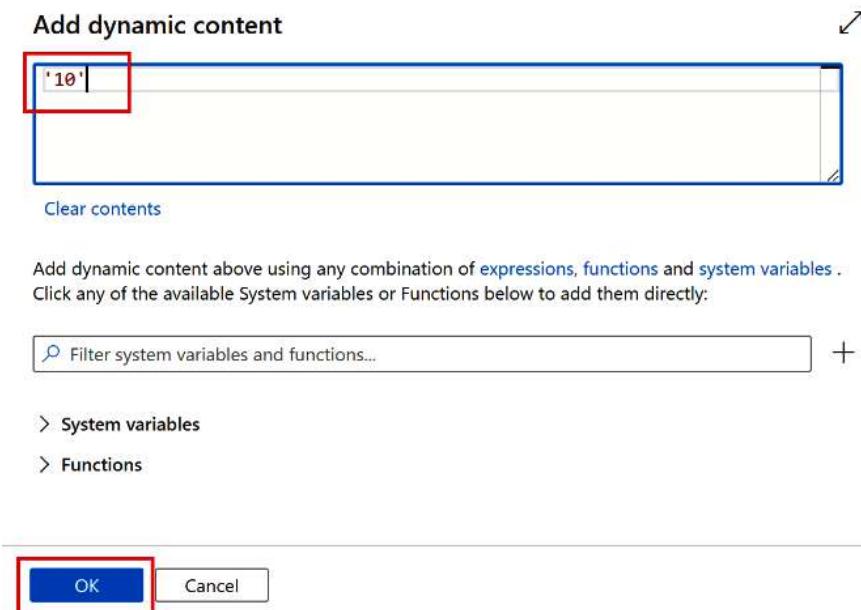


Figure 9.51 – Passing a value to the parameter

7. Publish the pipeline, click on **Add trigger**, and select **Trigger now** to execute the pipeline, as done in *step 17* of the *How to do it...* section in the *Copying data using a Synapse data flow* recipe.
8. Once the run completes, click on the monitoring icon on Synapse Studio. Select the latest run of the **Copy_Data_Flow** pipeline. Click on the **Dataflow1** activity details. Refer to *steps 1 to 4* of the *How to do it...* section in the *Monitoring data flows and pipelines* recipe for detailed example screenshots for a similar task. Notice that the pipeline only processed **28957** rows, compared to 300,000 rows initially present in the `transaction_table-t1.csv` file, as the rows were filtered by parameter:

The screenshot shows the Microsoft Azure Synapse Analytics studio interface. On the left, there's a sidebar with various options like Analytics pools, Activities, and Integration. The 'Integration' section has 'Pipeline runs' selected, which is highlighted with a red box. The main area displays the details for a 'Data flow1' activity. It shows a cluster startup time of 2m 27s, 3 transformations, and a success status. Below this, a diagram shows three stages: a blue cylinder (sink), an orange funnel (filter transformation), and a blue battery (source). Under the 'Sinks' tab, a table lists the sink details. The table has columns: Sink, Status, Processing time, Highest processing..., Rows written, and Stages. There is one row for Parquet, which is marked as succeeded with a green checkmark, took 3s 441ms, and wrote 2s 728ms. The 'Rows written' column shows the value 28957, which is also highlighted with a red box. The 'Stages' column shows a single stage icon.

Sink	Status	Processing time	Highest processing...	Rows written	Stages
Parquet	✓ Succeeded	3s 441ms	2s 728ms	28957	1

Figure 9.52 – The transfer of the filtered rows

How it works...

We added a parameter named `sid` to the **Copy_CSV_to_Parquet** data flow. We wrote an expression in the filter transformation in the data flow that compared the `sid` parameter with the `sid` column in the data flow. The comparison expression in the filter transformation filtered the rows based on the condition that we defined. We passed the value to the `sid` data flow parameter from the **Copy_Data_Flow** pipeline's data flow activity. The data flow activity exposed the parameters that were present inside the data flow to the pipeline. We passed the value '`10`' from the **Copy_Data_Flow** pipeline to the data flow using the pipeline's data flow activity and all the rows that had a value equal to `10` on the `sid` column were selected and copied in Parquet format to the sink destination.

Handling schema changes dynamically in data flows using schema drift

A common challenge in **extraction, transformation, and load** (ETL) projects is when the schema changes at the source and the pipelines that are supposed to read the data from the source, transform it, and ingest it to the destination, start to fail. Schema drift, a feature in data flows, addresses this problem by allowing us to dynamically define the column mapping in transformations. In this recipe, we will make some changes to the schema of a data source, use schema drift to detect the changes, and handle changes without any manual intervention gracefully.

Getting ready

Create a Synapse Analytics workspace as explained in the *Provisioning an Azure Synapse Analytics workspace* recipe in *Chapter 8, Processing Data Using Azure Synapse Analytics*.

Complete the *Copying data using a Synapse data flow* recipe in this chapter.

How to do it...

In this recipe, we will be using the **Copy_CSV_to_Parquet** data flow completed in *Copying data using a Synapse data flow*. We will perform the following tasks:

- Add columns to a CSV file used as the source in the **Copy_CSV_to_Parquet** data flow from the *Copying data using a Synapse data flow* recipe
- Observe the new columns that are added being detected by the data flow using the schema drift feature
- Split the originally expected columns and newly detected columns into two different streams
- Use rule-based column mapping to dynamically map the new detected columns

Perform the following steps to achieve this:

1. Download the `transaction_table-t1.zip` file from <https://github.com/PacktPublishing/Azure-Data-Engineering-Cookbook-2nd-edition/upload/main/chapter9> and unzip it. Open the `transaction_table-t1.csv` file on your machine using Excel.
2. Insert a column named `d1` after the `pid` column but before the `c1` column. Fill some random values in for the first 10 rows. Insert a column named `d2` after the last column, `c2`. Fill some random values in for the first 10 rows. Save the file:

tid	transaction	order_cour	total_cost	sid	pid	d1	c1	c2					d2
1	20211111	31	44620	1	9	Xyz	FE220CAF-81FBBA6D-384A-45CD-A5A7-04E558FFAE8C						abc
2	20211001	39	30518	2	8	Xyz	E75DAF40-7958C45C-F8E4-4E64-A485-B38820E14703						abc
3	20220315	74	8095	3	7	Xyz	898B1D41-5D9F3A2E-7FBF-4361-8965-D1A6FD5B1BC						abc
4	20211009	6	4658	4	6	Xyz	52913889-18F854DF-E633-4210-B7C2-BB21D86B7012						abc
5	20210609	36	27711	5	5	Xyz	F9FE0972-7748D5F21-C02F-4587-9A88-FA8E7F114DFD						abc
6	20210609	66	1477	6	4	Xyz	41D64EDD-7FD92194-2FEF-4B7E-AEC4-D2A93E26BEFD						abc
7	20210930	2	9254	7	3	Xyz	C4127574-E8B9E923-7D18-4B3F-8AA5-32685FCEBE56						abc
8	20210916	10	2418	8	2	Xyz	9A4D87D7-1AB787C4-A214-4EB8-9EA0-0CF4618B03E8						abc
9	20211216	22	17981	9	1	Xyz	B7B10CF9-792B36E1-989A-4851-AB89-AD8B1E56CF3D						abc
10	20211220	25	41993	1	10	Xyz	361F6444-12E25E5F4-D299-4C8E-8E99-C5C21B46AE5C						abc
11	20210914	24	44402	2	9		93E195C6-89E6B319-4BD6-4303-BB2C-6DC8C412CA16						
12	20220325	55	5399	3	8		5C23E159-BCE0E87CE-6189-482F-A31B-C9790209438B						
13	20220227	70	26656	4	7		0314C1FB-2400BB22-E57B-43BE-B543-A685A3C27C79						

Figure 9.53 – Adding new columns to the data source

- Upload the `transaction_table-t1.csv` file to the `synapse/csv` folder in the storage account linked with the Synapse workspace. Ensure that you overwrite the existing file. For detailed example screenshots for a similar task, follow steps 1 to 4 in the *How to do it...* section from the *Analyzing data using serverless SQL pool* recipe of *Chapter 8, Processing Data Using Azure Synapse Analytics*.
- Within `packtadesynapse` of Synapse Studio, click on the **Develop** icon (the notebook-like symbol) on the left-hand side of the screen. Expand **Data flows** and click **Copy_CSV_to_Parquet**. Turn on the debug mode by clicking on the toggle button at the top:

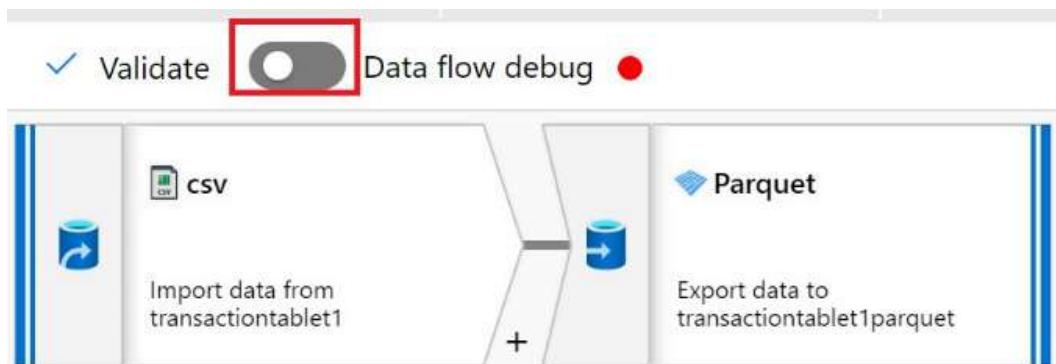


Figure 9.54 – Turning on the debug mode

5. Click **OK** to enable the debug mode for the data flow. We will use the debug mode to see how data flows through the transformations in this recipe:

Turn on data flow debug

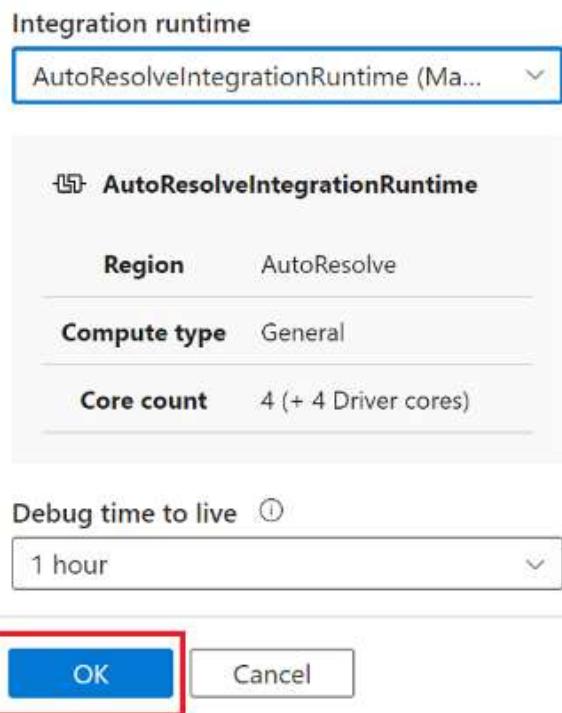


Figure 9.55 – Turning on the debug mode

6. Click on source transformation (**csv**). Observe that **Allow schema drift** is turned on by default in **Source settings**:

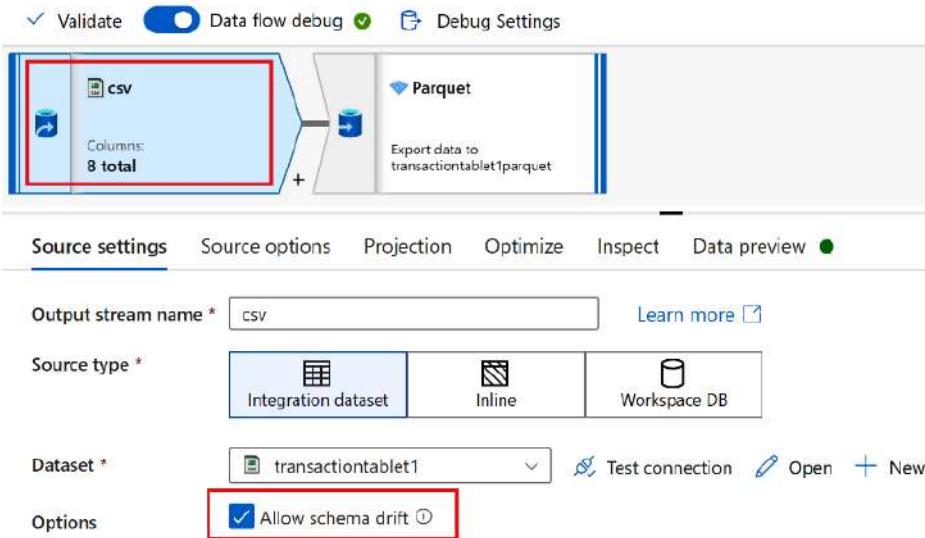


Figure 9.56 – Schema drift is turned on by default

- Once the debug mode is ready (it typically takes 5 minutes to prepare after turning on), go to **Data preview**. Hit the **Refresh** button. Scroll to the right. Notice that the new columns we added in the Excel sheet, namely d1 and d2, have been automatically added. Ignore the _c9, _c10, and _c11 columns that Excel has added by itself. Also, notice that even though we added the d1 column in the middle of the existing columns, the data flow has positioned it to the end, as it is not an expected column and has been newly added. Click on the **Map drifted** button:

The screenshot shows the 'Data preview' tab for the same data flow. At the top, there are status indicators: 'Validate' (green checkmark), 'Data flow debug' (blue toggle switch), and 'Debug Settings'. Below these are the two data nodes: 'csv' source and 'Parquet' sink. The 'Data preview' tab is selected, indicated by a red box around its tab header. Underneath are various data manipulation buttons: 'Refresh' (highlighted with a red box), 'Typecast', 'Modify', 'Map drifted' (highlighted with a red box), 'Statistics', 'Remove', 'Export CSV', and a 'Description' button. The data preview table has 10 columns: c2, abc, d1, abc, _c9, abc, _c10, abc, _c11, abc, and d2. The first four columns (c2, abc, d1, abc) have red boxes around them. The data rows show values for each column across five rows of data. The last two columns (abc, d2) are empty.

Figure 9.57 – Detecting the new columns

8. Clicking the **Map drifted** button creates a derived column transformation. Click on it and notice that the data flow has automatically created the mappings for the new columns. To make the column mapping dynamic, let's delete the existing column mappings first. Select all the column mappings by checking the checkboxes in the **Columns** list and hitting **Delete**:

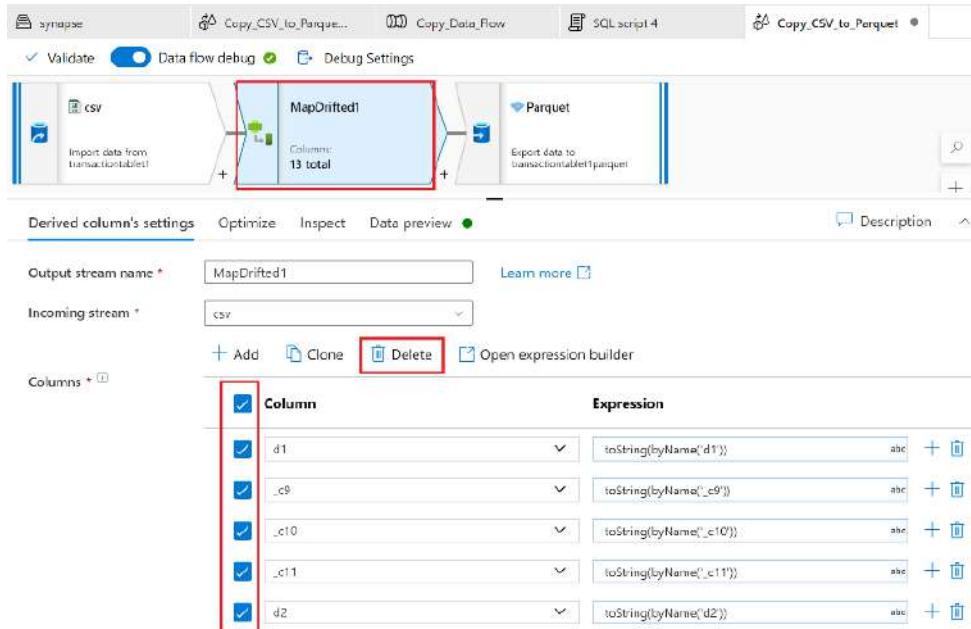


Figure 9.58 – Deleting the column mappings

9. To make column mapping dynamic, let's use the column pattern option. Click **Add** and select **Add column pattern**:

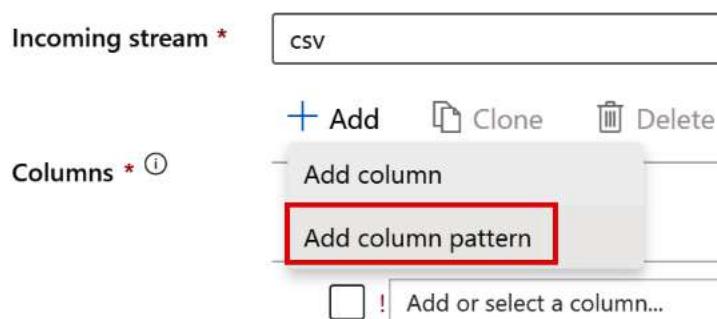


Figure 9.59 – Adding a column pattern

10. Click on the **column1** mapping and hit the delete button. Check the checkbox next to **Each column that matches** and type in **position > 8**. Type **\$\$** in both textboxes, on the left and right. By specifying **position > 8**, for each column after position number 8 in the list of columns read by the **Mapdrifted1** transformation, the data flow will dynamically create a new column. As all our columns after position 8 (counting the columns from left to right) are derived columns (new columns), we have configured the data mapping to automatically create columns for them. **\$\$** represents the value of the column and by specifying **\$\$** in the left- and right-hand text boxes, we are accepting the values in the columns as they are:

The screenshot shows the 'Column' section of the Azure Synapse Dataflows interface. At the top, there are buttons for 'Add', 'Clone', 'Delete', and 'Open expression builder'. Below these, the 'Column' section is displayed with the following configuration:

Column	Expression
column1	Enter expression... ANY
Each column that matches position > 8	creates 1 column(s) ANY
\$\$	abc
\$\$	ANY

Red boxes highlight several key elements: the 'Each column that matches' checkbox, the 'position > 8' condition, and the '\$\$' placeholder in both the left and right expression fields.

Figure 9.60 – Adding a column pattern

11. A key objective is to separate the derived columns and originally mapped columns into two streams so that the original data transformation from source to sink doesn't break and the expected columns are transferred as planned from the CSV file to the Parquet destination. To achieve this, we can configure a fixed rule mapping for the **Parquet sink transformation**. Click on the **Parquet** sink transformation and go to the **Mapping** section. The original columns are automatically listed. Check the checkbox next to **Input columns** to map all eight original columns that are expected to be copied to the Parquet destination folder:

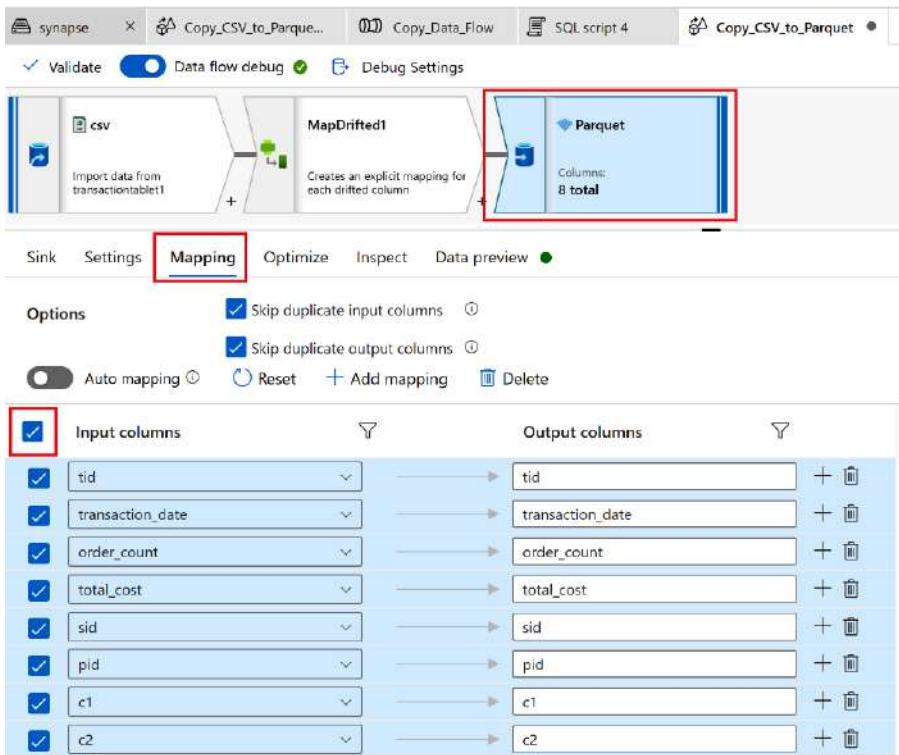


Figure 9.61 – The Parquet sink destination

- Now, let's get the primary key and the drifted columns into a separate stream. Click on the + symbol on the right-hand side of the **MapDrifted1** transformation again and add **Sink**. This sink will serve as the secondary path for tracking whether there are any drifted columns from the source:



Figure 9.62 – Adding a second sink

13. For the purpose of testing, we set **Sink type** to **Cache**. You can set the sink for the drifted columns to a Parquet dataset (or any integration dataset) if you wish to store the values of the drifted column:

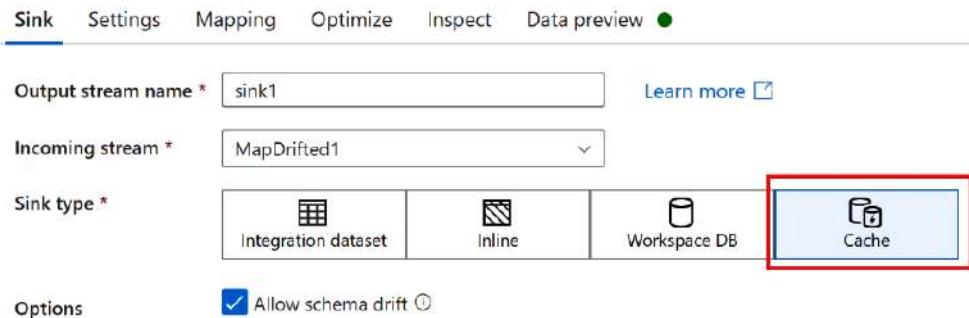


Figure 9.63 – The second sink type

14. Click on the newly added **sink1** transformation (the cache sink) and go to the **Mapping** section. Disable **Auto mapping**. Delete all mapping except the mapping for the **tid** column, as shown in the following screenshot. Once done, click **Add mapping** and select **Rule-based mapping**. Rule-based mapping helps us to dynamically create and define columns based on the column patterns:

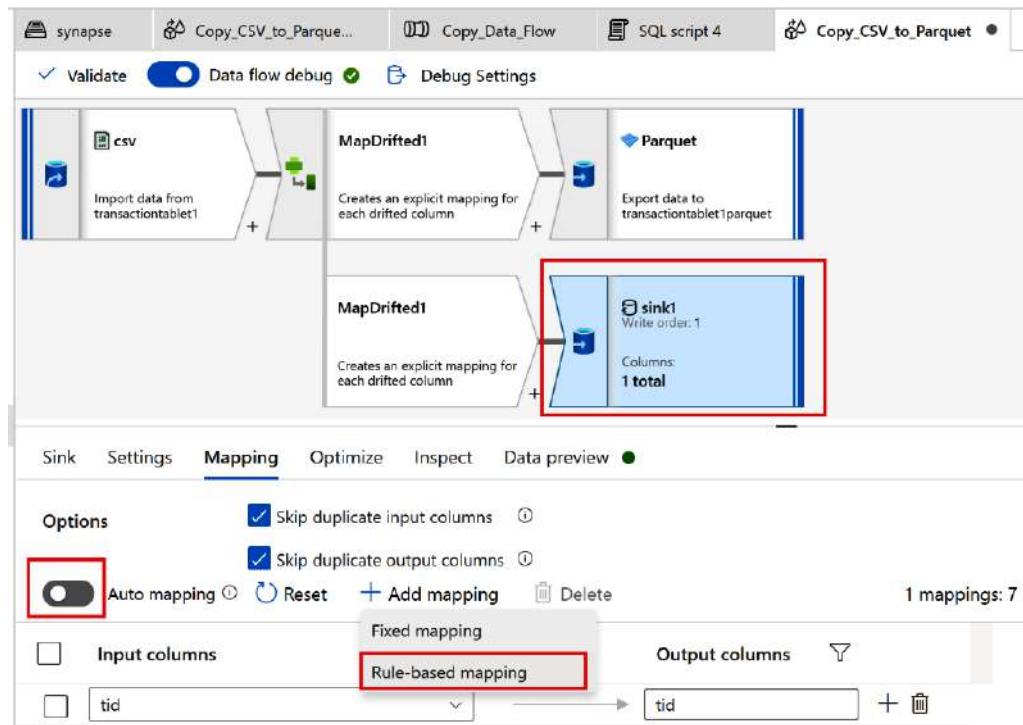


Figure 9.64 – Rule-based mapping

15. For **Rule-based mapping**, specify the column pattern as **position > 8** and the value as **\$\$**, which implies that columns from any position after 8 will be used as they are, as in **step 10**. Check the **tid** mapping and the **Rule-based mapping** checkboxes:

Sink Settings **Mapping** Optimize Inspect Data preview

Options

- Skip duplicate input columns
- Skip duplicate output columns

Auto mapping Reset 2 mapping

Input columns	Output columns
tid	tid
position > 8	\$\$

Figure 9.65 – Rule-based mapping for the sink

16. Click **Data preview** and hit the **Refresh** button to see the **tid** column and the derived columns only:

Sink Settings Mapping Optimize Inspect **Data preview**

Number of rows: 6 N/A N/A N/A N/A N/A N/A

tid	abc	d1	abc	d3	abc	c9	abc	c10	abc	c11	abc	d2	abc
1		Xyz		NULL		NULL		NULL		NULL		abc	
2		Xyz		NULL		NULL		NULL		NULL		abc	
3		Xyz		NULL		NULL		NULL		NULL		abc	
4		Xyz		NULL		NULL		NULL		NULL		abc	
5		Xyz		NULL		NULL		NULL		NULL		abc	
6		Xyz		NULL		NULL		NULL		NULL		abc	

Figure 9.66 – The result at the sink

How it works...

Schema drifting and rule-based mapping offer resilience and flexibility for the pipelines to accommodate almost any kind of change made at the data source. In this recipe, we used the **Map drifted** columns option and a derived columns transformation in *step 7* and *step 8* to identify the unplanned or the new columns for the source. We split the new columns using rule-based mapping to flow to a separate sink (a cache sink for testing) and ensured that the originally expected columns flowed to the Parquet destination using fixed rule mapping. Using the preceding recipe, data engineers can ensure that their pipelines transfer data as expected, but they can also track any new columns that are added to the source and can make adjustments to their pipelines if required.

10

Building the Serving Layer in Azure Synapse SQL Pool

As introduced in *Chapter 8, Processing Data Using Azure Synapse Analytics*, Azure Synapse Analytics comprises three key components – the Synapse integration pipeline to ingest and transform the data, the SQL pool and the Spark pool to process and serve the data, and Power BI integration to visualize the data. The SQL pool, the relational data warehouse engine, is usually the layer that maintains the processed data and will be used as the data serving layer consumed by reporting solutions. So, this chapter will focus on how to load processed data into a dedicated SQL pool and maintain dedicated SQL pools.

In this chapter, we will cover the following main topics:

- Loading data into dedicated SQL pools using PolyBase and T-SQL
- Loading data into dedicated SQL pools using COPY INTO
- Creating distributed tables and modifying table distribution
- Creating statistics and automating the update of statistics
- Creating partitions and archiving data using partitioned tables
- Implementing workload management in an Azure Synapse dedicated SQL pool
- Creating workload groups for advanced workload management

By the end of the chapter, you will have learned how to load data into a SQL pool using the COPY INTO and PolyBase methods, distribute the table using hash/replicate/round-robin algorithms, partition and archive data, maintain table statistics, and perform efficient resource allocation and management using workload management.

Technical requirements

For this chapter, you will need the following:

- A Microsoft Azure subscription

Loading data into dedicated SQL pools using PolyBase and T-SQL

In this recipe, we will load a CSV file into a dedicated SQL pool using PolyBase. PolyBase technology involves creating an external table that links the SQL pool with the file(s) stored in data lake storage. Once the PolyBase table has been created, you will be able to query the external table like any other table and data will be returned from data lake storage seamlessly. This recipe will involve the following tasks:

- Creating a dedicated Synapse SQL pool
- Creating an external table to read CSV files
- Loading the data read from an external table into a regular table stored in a Synapse dedicated SQL pool

Getting ready

To get started, perform the following steps:

1. Log in to <https://portal.azure.com> using your Azure credentials.
2. Create a Synapse Analytics workspace as explained in the *Provisioning an Azure Synapse Analytics workspace* recipe of *Chapter 8, Processing Data Using Azure Synapse Analytics*.
3. Download the `transaction-tbl.csv` and `transaction-tbl.parquet` files from <https://github.com/PacktPublishing/Azure-Data-Engineering-Cookbook-2nd-edition/tree/main/chapter10>.
4. In the Synapse Analytics workspace, create a folder named `files` in the data lake account attached to the Synapse Analytics workspace. Upload the `transaction-tbl.csv` and `transaction-tbl.parquet` files to the `files` folder.

For detailed screenshots from a similar task, follow *steps 1 to 4* in the *How to do it...* section from the *Analyzing Data Using Serverless SQL Pool* recipe of *Chapter 8, Processing Data Using Azure Synapse Analytics*.

How to do it...

Perform the following steps to load the data:

1. First, let's create a Synapse dedicated SQL pool. Log in to [portal.azure.com](#), go to **All resources**, and search for `packtadesynapse`, which is the Synapse Analytics workspace that was created in the *Provisioning an Azure Synapse Analytics workspace* recipe of *Chapter 8, Processing Data Using Azure Synapse Analytics*. Click on the workspace. Click on **Open Synapse Studio**:

The screenshot shows the Azure portal interface for the 'packtadesynapse' Synapse workspace. The left sidebar contains navigation links for Overview, Activity log, Access control (IAM), Tags, Diagnose and solve problems, Settings (Azure Active Directory, Properties, Locks), Analytics pools (SQL pools, Apache Spark pools, Data Explorer pools (preview)), and Security (Encryption). The main content area displays workspace details under 'Essentials': Resource group (move) : [PacktADESynapse](#), Status : Succeeded, Location : (empty), Subscription (move) : [Visual Studio Ultimate wit](#), Subscription ID : (empty), Managed virtual network : Yes, Managed Identity object ... : (empty), Workspace web URL : <https://web.azuresynapse.>, and Tags (edit) : [Click here to add tags](#). Below this is a 'Getting started' section with a red-bordered box containing the 'Open Synapse Studio' button, the Synapse logo, and the text: 'Start building your fully-integrated analytics solution and unlock new insights.'

Figure 10.1 – Open Synapse Studio

2. Click on the manage button (the briefcase symbol) on the left-hand side, which will take you to the **Manage** section. Under **SQL pools**, click on **New**:

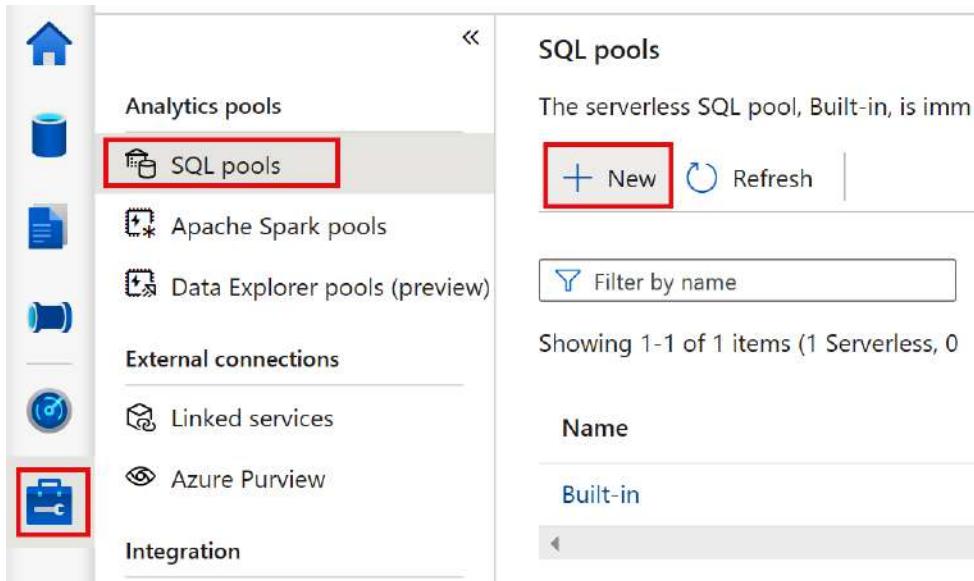


Figure 10.2 – Creating a new SQL pool

3. Create a dedicated SQL pool called `packtadesqlpool`. Set **Performance level** to **DW100c**. Click on **Review + create** and create the SQL pool:

New dedicated SQL pool

Basics * Additional settings * Tags Review + create

Create a dedicated SQL pool with your preferred configurations. Complete the Basics tab then go to Review + Create to provision with smart defaults. [Learn more](#)

Dedicated SQL pool details

Name your dedicated SQL pool and choose its initial settings.

Dedicated SQL pool name *

packtadesqlpool

Performance level ⓘ

DW100c

Estimated price ⓘ

Est. cost per hour

1.51 USD

[View pricing details](#)

[Review + create](#)

[Next: Additional settings >](#)

Figure 10.3 – Provisioning the SQL pool

- Click on the blue cylinder (the data symbol) on the left-hand side, which will take you to the **Data** section. Click on the **Linked** tab. Expand the data lake account of the Synapse workspace (in this example, packtadesynapse). Click on the **synapse (Primary)** container. Open the **Files** folder. Right-click on the **transaction-tbl.csv** file. Select **New SQL script** and then select **Create external table**:

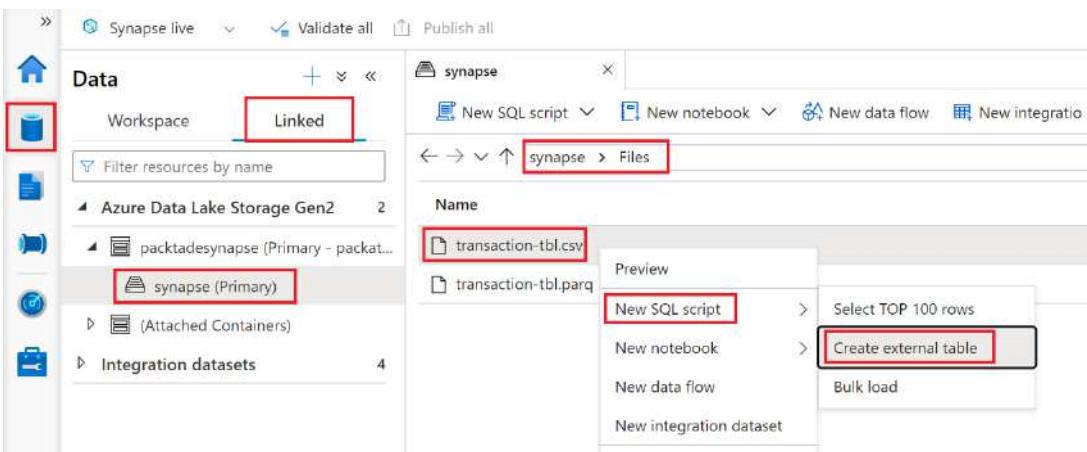


Figure 10.4 – Creating an external table

5. Select the **Infer column names** option. Click on **Continue**:

New external table

Source file format settings
Specify the format and layout of your data. [Learn more](#)

File path
synapse/Files/transaction-tbl.csv

Preview Data

File type
CSV

Field terminator ⓘ
Default (comma ,)
 Edit

First row ⓘ
0

Infer column names ⓘ

String delimiter ⓘ
Default (Empty string)
 Edit

Use default type ⓘ
Default type (true,false)

Max string length * ⓘ
4000

Continue

Figure 10.5 – Configuring the external table

6. Select **packtadesqlpool** as the SQL pool where the external table will be created. Provide a table name of **ext_transaction_tbl**. Click on **Open script**, and the script for the external table will be generated:

New external table

Select target database

[Learn more](#)

Select SQL pool*

packtadesqlpool

Select a database*

packtadesqlpool

External table name

ext_transaction_tb

Create external table

Automatically Using SQL script

This will generate a SQL script and you will be required to run the SQL script.

Open script

Back

Cancel

Figure 10.6 – Generating the external table script

7. In the CREATE EXTERNAL FILE section, after the USE_TYPE_DEFAULT = FALSE line, hit *Enter* and add , FIRST_ROW = 2. This is to ignore the first row since it contains the column names:

```
1 IF NOT EXISTS (SELECT * FROM sys.external_file_formats WHERE name = 'SynapseDelimitedTextFormat')
2     CREATE EXTERNAL FILE FORMAT [SynapseDelimitedTextFormat]
3     WITH (FORMAT_TYPE = DELIMITEDTEXT ,
4           FORMAT_OPTIONS (
5               FIELD_TERMINATOR = ',',
6               USE_TYPE_DEFAULT = FALSE
7               ,FIRST_ROW = 2
8           )
9     GO
```

Figure 10.7 – Adjustment for the first row

8. In the CREATE EXTERNAL TABLE section, change the data type for columns c1 and c2 to varchar(200) as the uniqueidentifier data type is not yet supported on SQL pools:

```

CREATE EXTERNAL TABLE ext_transaction_tbl (
    [tid] bigint,
    [transaction_date] bigint,
    [order_count] bigint,
    [total_cost] bigint,
    [sid] bigint,
    [pid] bigint,
    [c1] varchar(200),
    [c2] varchar(200)
)
WITH (
    LOCATION = 'Files/transaction-tbl.csv',
    ...
)
  
```

Figure 10.8 – Data type adjustment

9. Hit the Run button and verify whether the data has been read successfully from the external table:

The screenshot shows the Azure Data Studio interface with several tabs open. The 'Run' tab is highlighted with a red box. The code pane contains the following SQL script:

```

1 IF NOT EXISTS (SELECT * FROM sys.external_file_formats WHERE name = 'SynapseDelimitedTextFormat')
2     CREATE EXTERNAL FILE FORMAT [SynapseDelimitedTextFormat]
3         WITH (FORMAT_TYPE = DELIMITEDTEXT ,
4               FORMAT_OPTIONS (
5                   FIELD_TERMINATOR = ',',
6                   USE_TYPE_DEFAULT = FALSE
7                   ,FIRST_ROW = 2
8               ))
9
10 GO
11
12 IF NOT EXISTS (SELECT * FROM sys.external_data_sources WHERE name = 'synapse_packatadesynapse_dfs_core_windows_net')
13     CREATE EXTERNAL DATA SOURCE [synapse_packatadesynapse_dfs_core_windows_net]
14         WITH (
15             LOCATION = 'abfss://synapse@packatadesynapse.dfs.core.windows.net',
16             TYPE = HADOOP
17         )
18 GO
19
20 CREATE EXTERNAL TABLE ext_transaction_tbl [
21     [tid] bigint,
  
```

The results pane shows the data from the external table:

tid	transaction_date	order_count	total_cost	sid	pid	c1	c2
182753	20220301	18	37389	2	9	BF01F77A-1CA...	0C1C10
248912	20220301	34	15401	3	8	EDAD7939-1D3...	98163C
21158	20220228	67	27433	10	1	9548BA3F-CD0...	59D52C
248936	20220327	2	3731	7	4	CC9C2896-56B...	E2C7C0
30734	20220326	49	26984	3	8	61153A95-8D3...	4BA18A

At the bottom of the results pane, a message indicates: 00:00:09 Query executed successfully.

Figure 10.9 – Connecting to Azure Data Lake Gen 2

10. Scroll to the bottom of the SQL script window and add the following lines to create a regular table in the dedicated SQL pool and to load the data into a regular table from the external table. The CREATE TABLE statement creates a regular table named dbo.transaction_tbl stored in a SQL pool. The CREATE TABLE statement creates a table with the structure and result set as returned by the SELECT query, which follows the AS clause. Hit the **Run** button:

```
CREATE TABLE dbo.transaction_
tbl WITH (DISTRIBUTION = ROUND_ROBIN)
AS
Select * from dbo.ext_transaction_tbl;
GO
Select TOP 100 * from dbo.transaction_tbl
GO
```

The script's result is shown in the following screenshot:

The screenshot shows the Azure Data Studio interface with the following details:

- Toolbar:** Includes tabs for synapse, SQL script 5, and SQL script 6. Buttons for Run (highlighted with a red box), Undo, Publish, Query plan, Connect to, and Use database (set to packtadesqlpool).
- Script Editor:** Shows a T-SQL script with numbered lines. Lines 36-41 show the creation of a temporary table #transaction_errors. Lines 42-49 show the creation of the permanent table dbo.transaction_tbl, its AS clause definition, and the loading of data from the external table.
- Output Window:** Displays the results of the executed script. The results tab is selected, showing a table with columns: tid, transaction_date, order_count, total_cost, sid, pid, and c1. The data is as follows:

tid	transaction_date	order_count	total_cost	sid	pid	c1
86649	20220325	11	8459	1	10	26D2E934-51A...
161017	20220325	67	42385	1	10	674A6D4A-FB6...
239948	20220325	7	42434	1	10	9705BA27-14B...

Figure 10.10 – The data loaded and result set returned

You will see that the data has been successfully loaded onto `dbo.transaction_tbl` in the SQL pool, and it is read from the regular SQL pool table.

How it works...

Right-clicking on the CSV file and using the `CREATE EXTERNAL TABLE` option generates most of the SQL script required for creating an external table. For creating external tables against files in a data lake, we need three objects to be created, which are listed here:

- **External File Format** – This configures the file format to be read, which is CSV in our case.
- **External data source** – This specifies which file to be read and where it is located.
- **External table** – This is an actual external table listing the column names and their data type.

Fortunately, all of the preceding objects are automatically handled gracefully by Synapse Studio. Once created, we use the **Create Table As (CTAS)** method to ingest data into a regular table in a SQL pool. `CREATE TABLE <tablename> WITH (DISTRIBUTION = <>) AS <SELECT Query>` creates a regular table stored in the dedicated SQL data pool, on the fly, following the structure of the result set of the `SELECT` query. In other words, if the select query contains five columns, a table with five columns matching the data type of the columns returned by the select query will be created. Additionally, the result set data is inserted into the SQL pool table.

While the external table acts as a link to any file(s) stored in the data lake storage account, it is not suitable for running complex queries supporting analytic workloads. Regular tables stored inside a dedicated SQL pool are highly optimized for analytics, and using external tables to ingest the data from a data lake to a dedicated SQL pool is one of the two (the other being using `COPY INTO`) fastest methods available to move data from data lakes to Synapse dedicated SQL pools.

Loading data into a dedicated SQL pool using `COPY INTO`

Loading data into a dedicated SQL pool using the `COPY INTO` T-SQL statement is a very fast and efficient method. In this recipe, we will load one CSV file and one Parquet file into two dedicated SQL pool tables.

Getting ready

To get started, perform the following steps:

1. Log in to <https://portal.azure.com> using your Azure credentials.
2. Create a Synapse Analytics workspace, as explained in the *Provisioning an Azure Synapse Analytics workspace* recipe of *Chapter 8, Processing Data Using Azure Synapse Analytics*.
3. Download the `transaction-tbl.csv` and `transaction-tbl.parquet` files from <https://github.com/PacktPublishing/Azure-Data-Engineering-Cookbook-2nd-edition/tree/main/chapter10>.

4. In the Synapse Analytics workspace, create a folder named `files` in the data lake account attached to the Synapse Analytics workspace. Upload the `transaction-tbl.csv` and `transaction-tbl.parquet` files to the `files` folder.

For detailed screenshots from a similar task, follow *steps 1 to 4* in the *How to do it...* section from the *Analyzing data using serverless SQL pool* recipe of *Chapter 8, Processing Data Using Azure Synapse Analytics*.

5. Create a Synapse dedicated SQL pool named `packtadesqlpool`, as described in *steps 1 to 3* in the *How to do it...* section of the *Loading data into dedicated SQL pools using PolyBase using T-SQL* recipe.

How to do it...

1. Let's load a CSV file into a dedicated SQL pool using the `COPY INTO` statement. Log in to `portal.azure.com`, go to **All resources**, and search for `packtadesynapse`, which is the Synapse Analytics workspace that we created in the *Provisioning an Azure Synapse Analytics workspace* recipe of *Chapter 8, Processing Data Using Azure Synapse Analytics*. Click on the workspace. Click on **Open Synapse Studio**:

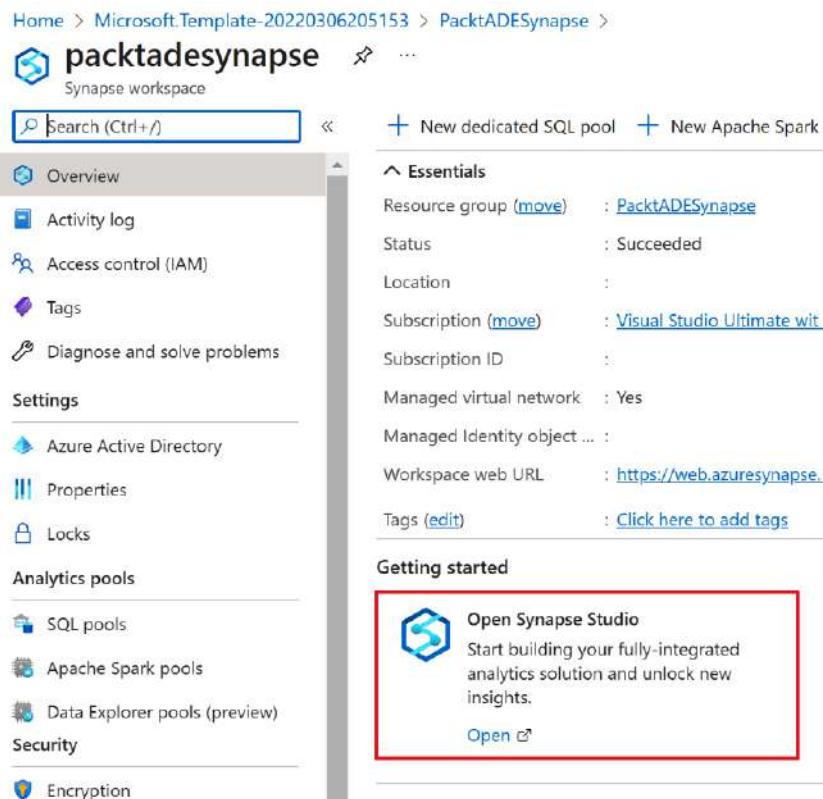


Figure 10.11 – Open Synapse Studio

2. Click on the develop button (the notebook-like symbol) on the left-hand side, which will take you to the **Develop** section. Click on the + button and select **SQL script**:

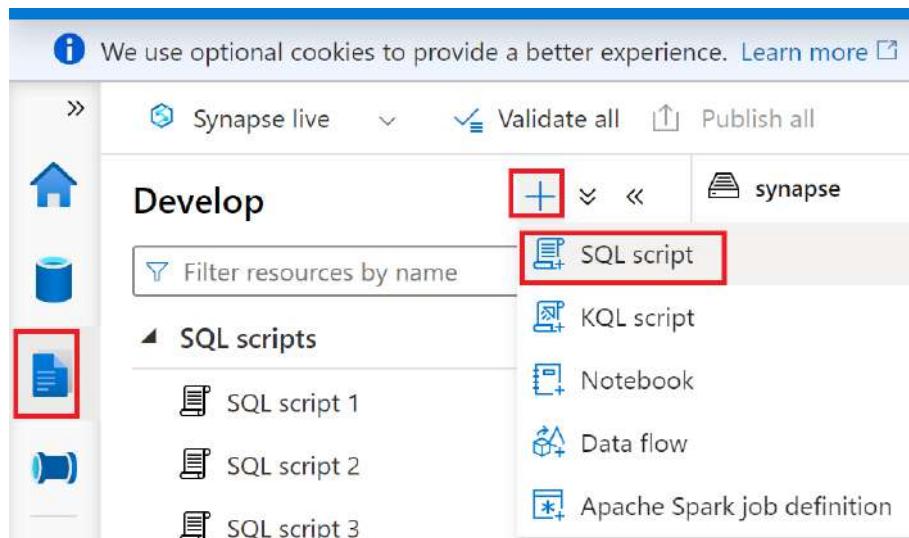


Figure 10.12 – SQL script

3. Paste the following script into the new SQL script window. Select **packtadesqlpool** in the **Connect to** option, and connect to **packtadesqlpool**:

```
CREATE TABLE dbo.transaction_tbl_copy([tid] bigint,
[transaction_date] date,
[order_count] bigint,
[total_cost] bigint,
[sid] bigint,
[pid] bigint,
[c1] nvarchar(200),
[c2] nvarchar(200))
WITH (  DISTRIBUTION  = ROUND_ROBIN);
GO
COPY INTO dbo.transaction_tbl_copy
FROM 'https://packtadesynapse.dfs.core.windows.net/
synapse/Files/transaction-tbl.csv'
WITH
(
FILE_TYPE = 'CSV',
```

```
MAXERRORS = 10,  
FIRSTROW = 2)  
Select top 100 * from dbo.transaction_tbl_copy
```

The preceding script creates a regular table, `transaction_tbl_copy`, in a dedicated SQL pool, `packtadesqlpool`. The `COPY INTO` statement loads the data from the `transaction-tbl.csv` file to the `transaction_tbl_copy` table. The select statement returns the rows by reading from the `transaction_tbl_copy` table. Update the path of `transaction-tbl.csv` as per your storage account's name and location, if required:

The screenshot shows the Azure Synapse Studio interface. The top navigation bar includes 'synapse', 'SQL script 5', 'Run' (highlighted with a red box), 'Undo', 'Publish', 'Query plan', 'Connect to' (highlighted with a red box and set to 'packtadesqlpool'), 'Use database' (set to 'packtadesqlpool'), and a dropdown for 'packtadesqlpool'. The main code editor area contains the following T-SQL script:

```
1 CREATE TABLE dbo.transaction_tbl_copy([tid] bigint,  
2     [transaction_date] date,  
3     [order_count] bigint,  
4     [total_cost] bigint,  
5     [sid] bigint,  
6     [pid] bigint,  
7     [c1] nvarchar(200),  
8     [c2] nvarchar(200))  
9     WITH ( DISTRIBUTION = ROUND_ROBIN);  
10    GO  
11  
12    COPY INTO dbo.transaction_tbl_copy  
13    FROM 'https://packatadesynapse.dfs.core.windows.net/synapse/Files/transaction-tbl.csv'  
14    WITH  
15    (  
16        FILE_TYPE = 'CSV',  
17        MAXERRORS = 10,  
18        FIRSTROW = 2)  
19  
20    Select top 100 * from dbo.transaction_tbl_copy  
21
```

The 'Results' tab is selected at the bottom, showing a table with the following data:

tid	transaction_date	order_count	total_cost	sid	pid	c1
222969	2022-03-29T00:00:00.000	35	11241	8	3	9B4B8630-6FC...
147675	2022-03-26T00:00:00.000	22	37325	4	7	E1D35594-76F...
433	2022-03-31T00:00:00.000	48	45838	5	6	4F804217-2625...
223590	2022-03-29T00:00:00.000	47	14696	5	6	8BC92A43-2FE...
640	2022-03-31T00:00:00.000	10	26423	3	8	FC6576AD-77B...

At the bottom, a message indicates the query was executed successfully.

Figure 10.13 – Loading data

4. Now, let's load the Parquet file. Copy the following script to the bottom of the query window to create a table in the dedicated SQL pool and load the data from the Parquet file. Select the following script and run it. Ensure to update the storage account name and location as per your environment:

```
COPY INTO dbo.transaction_tbl_Parquet
FROM 'https://packatadesynapse.dfs.core.windows.net/
synapse/Files/transaction-tbl.parquet'
WITH (
    FILE_TYPE = 'Parquet',
    AUTO_CREATE_TABLE = 'ON'
)
Select top 100 * from dbo.transaction_tbl_Parquet
```

The result of the data loading script is shown in the following screenshot:

The screenshot shows the Azure Data Studio interface. The top navigation bar includes 'synapse' (selected), 'SQL script 5', 'Run' (button highlighted with a red box), 'Undo', 'Publish', 'Query plan', 'Connect to' (set to 'packtadesqlpool'), 'Use database' (set to 'packtadesqlpool'), and a dropdown for 'Storage account'. The main code editor area displays a numbered script:

```
21
22
23 COPY INTO dbo.transaction_tbl_Parquet
24 FROM 'https://packatadesynapse.dfs.core.windows.net/synapse/Files/transaction-tbl.parquet'
25 WITH (
26     FILE_TYPE = 'Parquet',
27     AUTO_CREATE_TABLE = 'ON'
28 )
29
30
31 Select top 100 * from dbo.transaction_tbl_Parquet
```

The 'Results' tab is selected at the bottom, showing a table with the loaded data:

tid	transaction_date	order_count	total_cost	sid	pid	c1
31688	20211007	63	8860	1	10	74E453F0-D7D...
32284	20210826	74	1679	9	2	CE3519C3-228...
32880	20211229	30	42106	7	4	00D6B5F9-C70...
33476	20220124	4	43458	6	5	CA96D25C-7F1...
34072	20220117	2	33864	4	7	074B9C05-0EC...
34668	20220319	64	24180	2	9	541B7329-5525...

Figure 10.14 – Loading the Parquet file

A new table called `transaction_tbl_parquet` has been successfully created in the dedicated SQL pool and data was read successfully.

How it works...

`COPY INTO` is a simple, easy, and efficient way to load data into dedicated SQL pools. The `COPY INTO` method doesn't involve the creation of any additional objects such as external tables to load data into dedicated SQL pools.

While loading the data from the CSV file, we had to create the table and schema upfront and then issue the `COPY INTO` command to load the data. The schema defined in the script should exactly match the columns in the CSV file for the load to be successful. However, while loading the data to Parquet files, we didn't create a table up front, and we only issued one command, `COPY INTO`, to create the table in the dedicated SQL pool and load the data. We used the `AUTO_CREATE_TABLE = ON` option in the `COPY INTO` command while loading the Parquet files, as the `AUTO_CREATE_TABLE = ON` option is supported for Parquet files and it automatically creates the schema for the dedicated SQL pool table based on the Parquet file structure.

Creating distributed tables and modifying table distribution

A Synapse dedicated SQL pool is a distributed database allowing us to distribute data across multiple nodes and process it in parallel at multiple nodes to make data processing more efficient. To leverage the powers of the distributed database effectively, we need to store the data across the nodes with the correct strategy. A Synapse dedicated SQL pool offers three options while distributing the data, and they are listed as follows:

- **Round-Robin distribution** – This distributes the data equally across the nodes of the SQL pool.
- **Hash distribution** – This distributes the data based on the values of a column in the table.
- **Replicated** – This copies the table to all nodes of the dedicated SQL pool.

The table distribution option is selected at the time of table creation and can't be modified afterward. In this recipe, we will create a table specifying a distribution option, and we will modify the distribution option using a workaround.

Getting ready

To get started, perform the following steps:

1. Log in to <https://portal.azure.com> using your Azure credentials.
2. Create a Synapse Analytics workspace, as explained in the *Provisioning an Azure Synapse Analytics workspace* recipe of *Chapter 8, Processing Data Using Azure Synapse Analytics*.

3. Complete the *Loading data into dedicated SQL pools using PolyBase using T-SQL* recipe to create a dedicated SQL pool named packtadesqlpool, an external table named dbo.ext_transaction_tbl, and a dedicated SQL pool table named dbo.transaction_tbl.

Alternatively, you can use the Create_External_Table.SQL script from <https://github.com/PacktPublishing/Azure-Data-Engineering-Cookbook-2nd-edition/tree/main/chapter10> to create dbo.ext_transaction_tbl and dbo.transaction_tbl.

How to do it...

Perform the following steps to create and modify distributed tables:

1. Log in to portal.azure.com, go to **All resources**, and search for packtadesynapse. Click on the workspace. Click on **Open Synapse Studio**. Click on the develop button (the notebook-like button) on the left-hand side. Click on the + symbol and select **New SQL script**. Select packtadesqlpool in the **Connect to** option. Refer to *steps 1 and 2* in the *How to do it...* section of the *Loading data into dedicated SQL pools using COPY INTO* recipe from this chapter. In the new query window, copy and paste the following script:

```
CREATE TABLE dbo.transaction_tbl_
hash WITH (DISTRIBUTION = HASH(SID))
AS
Select * from dbo.ext_transaction_tbl;
GO
Select top 10 * from dbo.transaction_tbl_hash
```

The preceding script creates a table named dbo.transaction_tbl_hash, which is distributed using hash distribution. The DISTRIBUTION = HASH(SID) command in the WITH clause will create the distributed table based on the values within the SID column. The data from the external table, ext_transaction_tbl, is loaded into the dbo.transaction_tbl_hash table:

The screenshot shows the Azure Synapse Studio interface. On the left, there's a sidebar with icons for Home, Databases, SQL scripts (which is selected and highlighted with a red box), Notebooks, and Data flows. The main area has tabs for 'synapse' (selected), 'SQL script 5', and 'SQL script 6'. A toolbar at the top includes 'Run' (highlighted with a red box), 'Undo', 'Publish', 'Query plan', 'Connect to' (set to 'packtadesqlpool'), and 'Use database'. The 'SQL script 6' tab contains the following T-SQL code:

```

CREATE TABLE dbo.transaction_tbl_hash
WITH (DISTRIBUTION = HASH(SID))
AS
Select * from dbo.ext_transaction_tbl;
GO
Select top 10 * from dbo.transaction_tbl_hash

```

The results pane shows a table with columns: tid, transaction_date, order_count, total_cost, sid, and pid. The data is as follows:

tid	transaction_date	order_count	total_cost	sid	pid
4353	20220331	57	33358	10	1
6212	20220331	61	4690	6	5
515	20220331	14	32659	7	4

Figure 10.15 – Creating the hash table

2. Let's convert `dbo.transaction_tbl_hash` into replicate distribution. The steps involved in doing so are as follows:
 - Create a new table using the CTAS method and load the data from an existing table.
 - Drop the old table.
 - Rename the new table to the old table name.

Create a new query window, connect to `packtadesqlpool`, and paste the following script:

```

CREATE TABLE dbo.transaction_tbl_
temp WITH ( DISTRIBUTION = REPLICATE)
AS
SELECT * FROM dbo.transaction_tbl_hash;
DROP TABLE dbo.transaction_tbl_hash;
RENAME OBJECT dbo.transaction_tbl_temp to transaction_
tbl_hash
GO

SELECT TOP 10 * FROM dbo.transaction_tbl_hash
GO

```

The preceding script does the following:

- The CREATE TABLE command's WITH clause specifies DISTRIBUTION=REPLICATE as the option and is created using the CTAS syntax using a select query against the dbo.transaction_tbl_hash table. This ensures that the new table created (dbo.transaction_tbl_temp) is based on replicate distribution and it contains the same structure and data as dbo.transaction_tbl_hash.
- Once dbo.transaction_tbl_temp has been created with the same data as dbo.transaction_tbl_hash, the dbo.transaction_tbl_hash table can be dropped.
- Now dbo.transaction_tbl_temp is renamed to dbo.transaction_tbl_hash (the original table name) using the RENAME OBJECT command. The result is verified using the SELECT TOP 10 * command:

The screenshot shows the Azure Synapse Studio interface. At the top, there are tabs for 'synapse', 'SQL script 5', 'SQL script 6', and 'SQL script 7'. Below these are buttons for 'Run' (highlighted with a red box), 'Undo', 'Publish', 'Query plan', 'Connect to' (set to 'packtadesqlpool'), 'Use database' (set to 'pack'), and a search bar. The main area contains a SQL script with several numbered lines:

```

1 CREATE TABLE dbo.transaction_tbl_temp WITH (DISTRIBUTION = REPLICATE)
2 AS
3 SELECT * FROM dbo.transaction_tbl_hash;
4 DROP TABLE dbo.transaction_tbl_hash;
5 RENAME OBJECT dbo.transaction_tbl_temp TO transaction_tbl_hash
6 GO
7
8 SELECT TOP 10 * FROM dbo.transaction_tbl_hash
9 GO

```

Below the script, there are tabs for 'Results' (highlighted with a red box) and 'Messages'. Under 'View', the 'Table' tab is selected. The results grid displays the following data:

tid	transaction_date	order_count	total_cost	sid	pid	c1
269640	20220314	44	35619	10	1	28
234972	20220323	24	18456	10	1	88
145440	20220322	20	1312	10	1	75

Figure 10.16 – Changing the distribution

Using the preceding method, we have successfully changed the distribution of the table from hash distribution to replicate distribution.

How it works...

Setting the table distribution is a critical task and needs to be done at the time of table creation. To change the table distribution, we need to recreate the table with a new distribution option. In the preceding example, we created a new table based on old the table with a different distribution method, dropped the old table, and renamed the new table to the old name. While the preceding method works, in very large tables, this could be a resource-intensive task as we need to recreate the table again. So, getting the distribution strategy correct at the time of table creation can save a lot of time and effort. Secondly, as the method involves recreating the table and dropping the old table, any additional indexes, statistics, or permissions created in the old table need to be created again in the new table.

Creating statistics and automating the update of statistics

Statistics play a key role in maintaining the optimal performance of the queries in a Synapse dedicated SQL pool. For each query submitted, the query optimizer prepares a query plan that comprises several operations (such as filtering, joining, sorting, and more). Statistics inform the optimizer how many rows are expected to be returned for each of those operations, and based on the statistics input, the optimizer prepares query plans. So, for the query plans to be effective and performance to be optimal, we need good statistics.

In this recipe, we will learn how to create statistics and update statistics.

Getting ready

Create a Synapse Analytics workspace as explained in the *Provisioning an Azure Synapse Analytics workspace* recipe of *Chapter 8, Processing Data Using Azure Synapse Analytics*.

Complete the *Loading data into dedicated SQL pools using PolyBase using T-SQL* recipe to create a dedicated SQL pool named `packtadesqlpool`, an external table named `dbo.ext_transaction_tbl`, and a dedicated SQL pool table named `dbo.transaction_tbl`. Alternatively, you can use the `Create_External_Table.SQL` script from <https://github.com/PacktPublishing/Azure-Data-Engineering-Cookbook-2nd-edition/tree/main/chapter10> to create the `dbo.ext_transaction_tbl` and `dbo.transaction_tbl` tables.

How to do it...

In a Synapse dedicated SQL pool, by default, statistics are created automatically for tables stored in the SQL pool. For the external tables, the statistics are to be created manually. As the data in the table changes, the statistics are to be updated so that they can offer accurate estimates for the query optimizer to prepare efficient query plans. Statistics are not updated automatically for both external tables stored in a data lake and regular tables stored in Synapse dedicated SQL pools. So, in this recipe, we will perform the following tasks:

- Creating statistics for external tables
- Updating the statistics for external tables
- Updating the statistics for regular tables

The following steps can be used to perform these tasks:

1. First, let's create statistics against the external table, `dbo.ext_transaction_tbl`. Log in to `portal.azure.com`, go to **All resources**, and search for `packtadesynapse`. Click on the workspace. Click on **Open Synapse Studio**. Click on the develop button (the notebook-like button) on the left-hand side. Click on the + symbol and select **New SQL script**. Select `packtadesqlpool` in the **Connect to** option. Refer to *steps 1 and 2* in the *How to do it...* section of the *Loading data into dedicated SQL pools using COPY INTO* recipe for detailed instructions on how to open a new query window. In the new query window, copy and paste the script provided in the following snippet and hit the **Run** button:

```
CREATE STATISTICS ext_transaction_tbl_pid on ext_
transaction_tbl(pid)
GO
DBCC SHOW_STATISTICS(ext_transaction_tbl,ext_transaction_
tbl_pid)
GO
```

The `CREATE STATISTICS` command creates a statistic for the `pid` column in the `ext_transaction` external table. The `DBCC SHOW_STATISTICS` command reads the statistic and displays the expected number of rows for each value in the `pid` column and confirms that the statistic was successfully created. You can learn more about interpreting the results of `SHOW_STATISTICS` at <https://docs.microsoft.com/en-us/sql/t-sql/database-console-commands/dbcc-show-statistics-transact-sql?view=azure-sqldw-latest>:

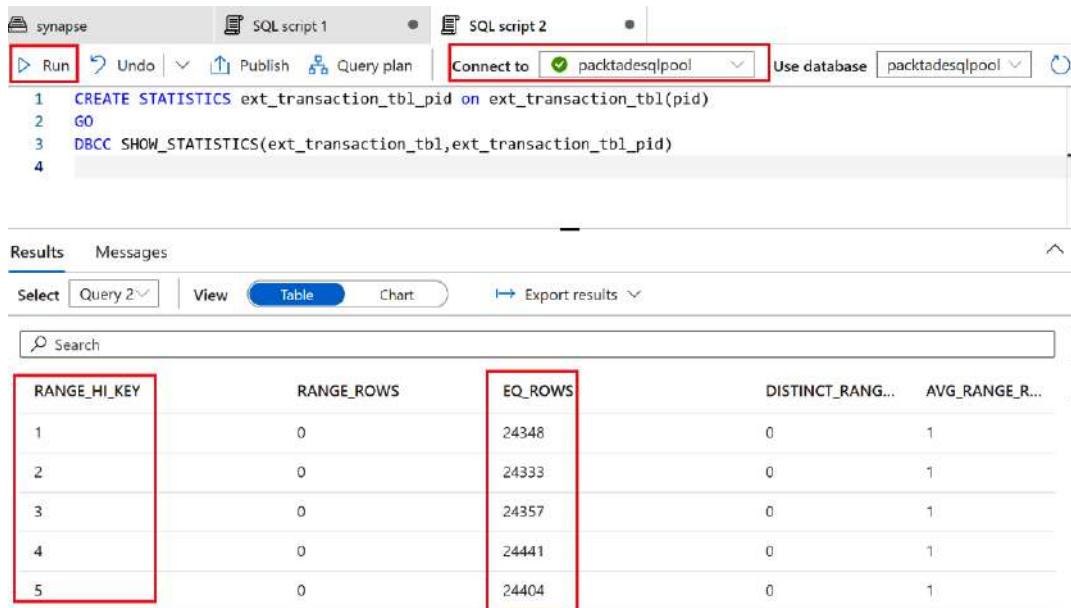
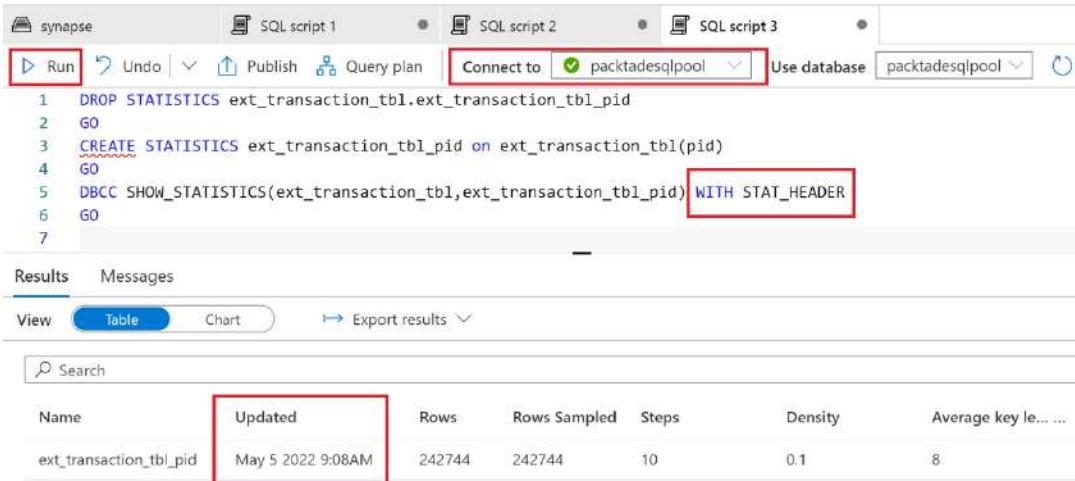


Figure 10.17 – Creating statistics in an external table

2. Let's try to update the statistic created for an external table. As of writing (May 2022), updating statistics is not supported for external tables. So, as a workaround, we will have to drop and create the statistic on an external table. In a new query window connected to the packtadesqlpool database, copy the following script and hit the **Run** button:

```
DROP STATISTICS ext_transaction_tbl.ext_transaction_tbl_
pid
GO
CREATE STATISTICS ext_transaction_tbl_pid on ext_
transaction_tbl(pid)
GO
DBCC SHOW_STATISTICS(ext_transaction_tbl,ext_transaction_
tbl_pid) WITH STAT_HEADER
GO
```

The `DROP STATISTICS` command specifies the table name and the statistic name that needs to be dropped. The `CREATE STATISTICS` command creates the statistic in the column again. The `DBCC SHOW_STATISTICS` command has a `WITH` clause specifying the `STAT_HEADER` option, which provides the date and time at which the statistic was updated or created:



The screenshot shows the Azure Data Studio interface. At the top, there are tabs for 'synapse', 'SQL script 1', 'SQL script 2', and 'SQL script 3'. Below these are buttons for 'Run' (highlighted with a red box), 'Undo', 'Publish', 'Query plan', 'Connect to' (set to 'packtadesqlpool'), 'Use database' (set to 'packtadesqlpool'), and a refresh icon. The main area contains a SQL script:

```

1 DROP STATISTICS ext_transaction_tbl.ext_transaction_tbl_pid
2 GO
3 CREATE STATISTICS ext_transaction_tbl_pid ON ext_transaction_tbl(pid)
4 GO
5 DBCC SHOW_STATISTICS(ext_transaction_tbl,ext_transaction_tbl_pid) WITH STAT_HEADER
6 GO
7

```

The 'WITH STAT_HEADER' part of the fifth line is highlighted with a red box. Below the script, there are tabs for 'Results' (selected) and 'Messages'. Under 'Results', there are buttons for 'View' (Table, Chart), 'Export results', and a search bar. The results table shows one row:

Name	Updated	Rows	Rows Sampled	Steps	Density	Average key le...
ext_transaction_tbl.pid	May 5 2022 9:08AM	242744	242744	10	0.1	8

Figure 10.18 – Updating the statistics in the external table

- As statistics are automatically created in regular tables in a dedicated SQL pool, let's focus on keeping the statistic updated. Download the `update_statistics_regular_table.SQL` file from https://github.com/PacktPublishing/Azure-Data-Engineering-Cookbook-2nd-edition/blob/main/chapter10/update_statistics_regular_table.sql. Click on the + button (the same button used to create a new query window) on the left-hand side and select **Import**:

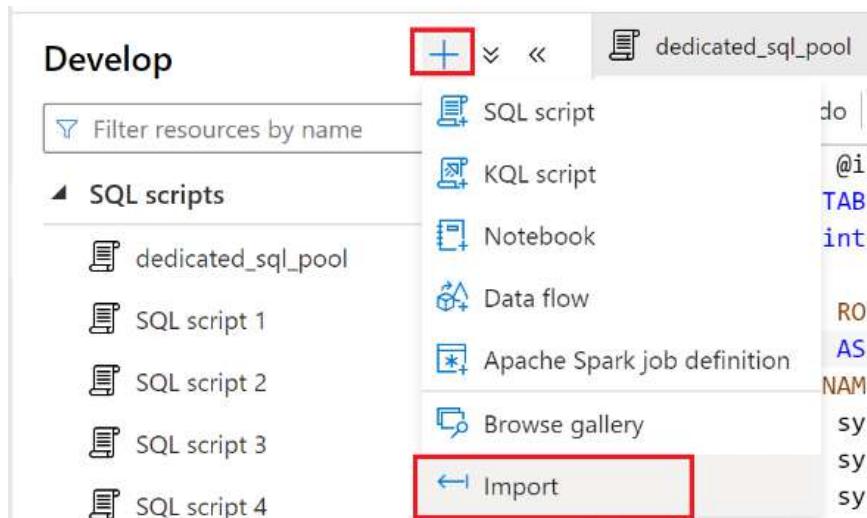


Figure 10.19 – Importing the SQL script

4. Select the downloaded `update_statistics_regular_table.SQL` script, connect to `packtadesqlpool`, and click on the **Run** button:

```

1 Declare @id int = 1,
2 @rowcount int = 0,
3 @query nvarchar(2500) CREATE TABLE #tmp(id int, full_stat_name varchar(2000))
4 Insert into #tmp
5 SELECT
6     ROW_NUMBER() OVER(ORDER BY (SELECT NULL)) AS [seq_nmbr],
7     QUOTENAME(sm.[name]) + '.' + QUOTENAME(tb.[name]) + '(' + QUOTENAME(st.[name]) + ')' as Full_stat_Name
8 FROM
9     sys.objects AS ob
10    JOIN sys.stats AS st ON ob.[object_id] = st.[object_id]
11    JOIN sys.tables AS tb ON st.[object_id] = tb.[object_id]
12    JOIN sys.schemas AS sm ON tb.[schema_id] = sm.[schema_id]
13 WHERE
14     DATEDIFF(
15         dd,
16         STATS_DATE(st.[object_id], st.[stats_id]),
17         GETDATE()
18     ) > 7
19 AND is_external = 0
20 Select @rowcount = count(*)
21 FROM
22     #tmp
23 WHILE @id <= @rowcount
24 BEGIN
25     Select @query = 'Update Statistics ' + Full_stat_Name
26     FROM #tmp
27     WHERE @id = id
28     EXEC sp_executesql @query
29     SET @id = @id + 1
30 END
31 DROP TABLE #tmp

```

Results Messages

00:00:04 Query executed successfully.

Figure 10.20 – Updating the statistics for regular tables

The script in `update_statistics_regular_table.SQL` identifies all the statistics in regular tables that have not been updated for the last 7 days. The script identifies them using system views, such as `sys.tables`, `sys.stats`, and a function called `STATS_DATE`, which returns the last date the statistic was updated.

The syntax for updating the statistic is `Update Statistic <Schema Name>. <Table Name>. <Statistic Name>`. The script checks whether the data returned by the `STATS_DATE` function is older than 7 days, prepares the `Update Statistic` command for each old statistic, and stores the command in a temporary table, `#tmp`. The script loops through each row in temporary table `#tmp` using a `WHILE` loop, extracts the `Update Statistic` command, and dynamically executes the `Update Statistic` command for each old statistic using the `sp_executesql` command.

How it works...

Creating statistics for external tables is straightforward. Updating statistics against external tables is not supported, and you might need to drop and recreate statistics to keep the statistics against the external table up to date. Additionally, we need to remember that for external tables, as statistics are not created automatically, we need to create them for the key columns (columns in the WHERE clause, JOIN conditions, sort operations, and GROUP BY clause) and maintain them diligently to keep the performance of the queries against external tables optimal.

For regular tables stored in a dedicated SQL pool, statistic creation is automatic by default. To keep the statistic updated, we used the script in the `update_statistics_regular_table.SQL` file. This script automatically identifies all the statistics in your database that haven't been updated for the last 7 days and updates each of the old statistics. The script could be created as a stored procedure and executed from the SQL pool stored procedure task in the Synapse integration pipeline and scheduled on a weekly basis to fix all outdated statistics.

Creating partitions and archiving data using partitioned tables

Partitioned tables split the table into smaller segments based on a particular column's values. Splitting the table into multiple partitions makes data loading and data archival faster and can also improve query performance. In this recipe, we will perform the following tasks:

- Creating a partitioned table and loading the data
- Archiving data older than 6 months using partition commands

Getting ready

Create a Synapse Analytics workspace as explained in the *Provisioning an Azure Synapse Analytics workspace* recipe of *Chapter 8, Processing Data Using Azure Synapse Analytics*.

Create a Synapse dedicated SQL pool named `packtadesqlpool` and an external table named `ext_transaction_tbl`, as described in steps 1 to 9 in the *How to do it...* section of the *Loading data into dedicated SQL pools using PolyBase using T-SQL* recipe. Alternatively, you can use the `Create_External_Table.SQL` script from <https://github.com/PacktPublishing/Azure-Data-Engineering-Cookbook-2nd-edition/tree/main/chapter10> to create the `dbo.ext_transaction_tbl` and `dbo.transaction_tbl` tables.

How to do it...

In this recipe, we will create a partitioned table called `Transaction_Partitioned`. We will create a partition based on the `transaction_date` column. We will partition the table into seven partitions in the following fashion:

- The first partition will contain transaction data from before October 2021.
- The second to sixth partitions will contain data from October 1, 2021, to March 1, 2022. Each partition will contain one month of data.
- The seventh partition will contain data after March 1, 2022.

Once the table has been partitioned as described in the preceding list, we will archive data (copy the data to another table and delete it from the original table) older than October 1, 2021, using a method called partition switching. The steps are provided as follows:

1. Download the `table_partition_boundary.sql` file from https://github.com/PacktPublishing/Azure-Data-Engineering-Cookbook-2nd-edition/blob/main/chapter10/table_partition_boundary.sql.
2. Log in to `portal.azure.com`, go to **All resources**, and search for `packtadesynapse`. Click on the workspace. Click on **Open Synapse Studio**. Click on the develop button (the notebook-like button) on the left-hand side. Click on the + symbol and select **Import**:

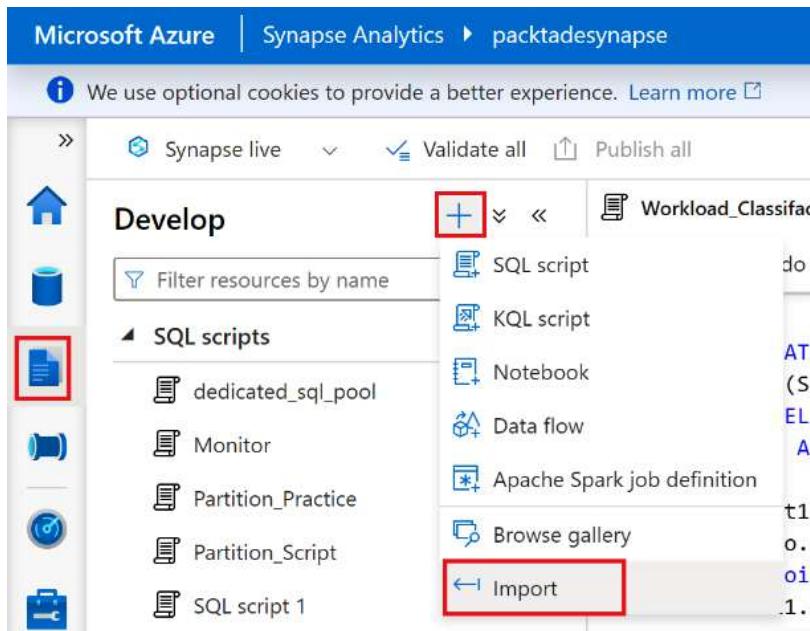


Figure 10.21 – Updating the statistics for the regular tables

3. Select the `table_partition_boundary.sql` file, which was downloaded in *step 1*. Connect to the `packtadesqlpool` database and run the script. The script creates a view called `table_partition_boundary`, which will be used in this recipe to get the table partition details:



The screenshot shows the Azure Synapse Studio interface with the following details:

- Top Bar:** Shows tabs for "Workload_Classification", "Monitor", and "table_partition_bou...".
- Toolbar:** Includes "Run", "Undo", "Publish", "Query plan", "Connect to", and "Use database". The "Connect to" dropdown is set to "packtadesqlpool" and is highlighted with a red box.
- Code Editor:** Displays the T-SQL script for creating the view `dbo.table_partition_boundary`. The first line of the script is also highlighted with a red box.
- Bottom Navigation:** Shows "Results" and "Messages" tabs, with "Results" currently selected.

```

1 CREATE VIEW dbo.table_partition_boundary
2 AS
3 Select t.name,rng.boundary_id, rng.value,ppt.rows
4 from sys.partitions ppt inner join sys.tables t
5 on ppt.object_id = t.object_id
6 INNER JOIN sys.indexes idx ON ppt.[object_id] = idx.[object_id]
7 AND ppt.[index_id] = idx.[index_id]
8 INNER JOIN sys.data_spaces ds ON idx.[data_space_id] = ds.[data_space_id]
9 INNER JOIN sys.partition_schemes ps ON ds.[data_space_id] = ps.[data_space_id]
10 INNER JOIN sys.partition_functions pf ON ps.[function_id] = pf.[function_id]
11 LEFT JOIN sys.partition_range_values rng ON pf.[function_id] = rng.[function_id]
12 AND rng.[boundary_id] = ppt.[partition_number]
13

```

Figure 10.22 – Creating the table partition boundary view

4. Let's create a partitioned table, `Transaction_Partitioned`, as described in the third step. Click on the + button and select **SQL Script** to open a new query window. Connect to `packtadesqlpool`. Copy and paste the following script into the query window and run the script:

```

CREATE TABLE [dbo].[Transaction_Partitioned]
WITH
(
    DISTRIBUTION = ROUND_ROBIN,
    CLUSTERED COLUMNSTORE INDEX,
    PARTITION ( [transaction_date] RANGE RIGHT FOR VALUES
                (20211001,20211101,20211201,20220101,
                20220201,20220301
                )
            )
AS

```

```
Select * from dbo.ext_transaction_tbl  
GO  
Select min(transaction_date) as min_trans_dt,  
       max(transaction_date) as max_trans_dt  
  from Transaction_Partitioned;  
GO
```

This script creates a partitioned table named `Transaction_Partitioned`. In the `Create table` command of the `Partition` clause, we have specified the `transaction_date` column, which implies that the table will be partitioned based on the `transaction_date` column. In the `RANGE RIGHT FOR VALUES` clause, six dates have been specified, which implies that the table will have seven partitions. The values specified in the `RANGE RIGHT FOR VALUES` clause are called **boundary values**. The first boundary value in the list is `20211001` (October 1, 2021), which would mean that all rows with `transaction_date` less than `20211001` (before October 1, 2021) will be stored in partition 1. The rows between `20211101` (November 1, 2021) and `20211201` (December 1, 2021) will be stored in partition 2 and similarly up till partition 6. All the rows after `20220301` (March 1, which is the last boundary value specified on the list) will be stored in partition 7. The script result is provided in the following screenshot:

The screenshot shows the SQL Server Management Studio interface. The top menu bar includes 'Workload Classificati...', 'Monitor', 'table_partition_boun...', 'Partition_Script', 'SQL script 5', 'Run', 'Undo', 'Publish', 'Query plan', 'Connect to', 'packtadesqlpool', 'Use database', 'packtadesqlpool', and a refresh icon.

The main area displays the following T-SQL code:

```
1 CREATE TABLE [dbo].[Transaction_Partitioned]  
2 WITH  
3 (  
4     DISTRIBUTION = ROUND_ROBIN,  
5     CLUSTERED COLUMNSTORE INDEX,  
6     PARTITION ( [transaction_date] RANGE RIGHT FOR VALUES  
7                 (20211001,20211101,20211201,20220101,20220201,20220301  
8                 )  
9             )  
10        )  
11    AS  
12 Select * from dbo.ext_transaction_tbl  
13 GO  
14 Select min(transaction_date) as min_trans_dt,max(transaction_date) as max_trans_dt  
15  from Transaction_Partitioned;  
16 GO
```

A red box highlights the boundary values in the `PARTITION` clause: `(20211001,20211101,20211201,20220101,20220201,20220301)`. A callout bubble points to the value `20220301` with the text "Partition 7 - Values greater than 20220301".

The results pane shows the output of the final `SELECT` statement:

min_trans_dt	max_trans_dt
20210801	20220331

Figure 10.23 – Creating the partition table

5. Let's check the `table_partition_boundary` view to see the partition details. Copy and paste the following query into the same query window and run the query to verify the partition:

```
Select * from dbo.table_partition_boundary
GO

14 Select min(transaction_date) as min_trans_dt,max(transaction_date) as max_trans_dt
15 from Transaction_Partitioned;
16 GO
17 Select * from dbo.table_partition_boundary
18 GO
```

name	boundary_id	value	rows
Transaction_Partitioned	1	20211001	34677
Transaction_Partitioned	2	20211101	34677
Transaction_Partitioned	3	20211201	34677
Transaction_Partitioned	4	20220101	34677
Transaction_Partitioned	5	20220201	34677
Transaction_Partitioned	6	20220301	34677
Transaction_Partitioned	(NULL)	(NULL)	34682

Figure 10.24 – Verifying the partition details

We can see seven rows representing seven partitions. The `value` column represents the boundary value and the `boundary_id` column is like the partition number. The seventh partition has `NULL` as its boundary value, as it contains all values greater than `20220301` and, hence, won't have a boundary. The `rows` column represents the approximate row count in each partition.

6. Let's move all the rows from partition 1 to another table as it contains rows from before October 1, 2021. To achieve that, we need to create an empty table with the exact same structure as our original table (`Transaction_Partitioned`). The empty table should be partitioned with a single boundary value that needs to be moved. Copy the following script to the same query window and run the query:

```
CREATE TABLE [dbo] . [Transaction_Partitioned_before_oct]
WITH
```

```
(  
    DISTRIBUTION = ROUND_ROBIN,  
    CLUSTERED COLUMNSTORE INDEX,  
    PARTITION ( [transaction_  
date] RANGE RIGHT FOR VALUES  
                (20211001  
                )  
            )  
        )  
AS  
Select * from dbo.Transaction_Partitioned where 1 = 2;
```

Here is the script in the query window:

```
19  CREATE TABLE [dbo].[Transaction_Partitioned_before_oct]  
20  WITH  
21  (  
22      DISTRIBUTION = ROUND_ROBIN,  
23      CLUSTERED COLUMNSTORE INDEX,  
24      PARTITION ( [transaction_date] RANGE RIGHT FOR VALUES  
25          (20211001  
26          )  
27      )  
28  )  
29  AS  
30  Select * from dbo.Transaction_Partitioned where 1 = 2;
```

Results Messages

Figure 10.25 – Creating an archival table

The CREATE Table command uses a select query from the Transaction_Partitioned table but with a WHERE clause, 1 = 2, to ensure only the structure of the table is copied into the new Transaction_Partitioned_before_oct table. Only the boundary value that needs to be archived is specified in the partition boundary value list of the new table.

7. Now, let's move partition 1 from the Transaction_Partitioned table to the Transaction_Partitioned_before_oct table. This can be done using the SWITCH command in the partition tables. Copy and paste the following commands into the same query window and run them:

```
ALTER TABLE Transaction_  
Partitioned SWITCH PARTITION 1 to Transaction_  
Partitioned_before_oct PARTITION 1
```

```

GO
Select name,boundary_id,value,rows
from dbo.table_partition_boundary
where value = 20211001
GO

```

The `ALTER TABLE` command moves partition 1 from `Transaction_Partitioned` to the `Transaction_Partitioned_before_oct` table's partition 1. For the move to be successful, three key conditions need to be met:

- The destination table and the source table should have the same structure.
- The boundary values of the partition being moved and the destination partition should match.
- The destination partition should be empty.

As we had met all the mentioned conditions, the move was successful. The move can be verified using a `Select` query against the `table_partition_boundary` view:

The screenshot shows a SQL query results window. The query is as follows:

```

31
32 ALTER TABLE Transaction_Partitioned SWITCH PARTITION 1 to Transaction_Partitioned_before_oct PARTITION 1
33 GO
34 Select name,boundary_id,value,rows
35 from dbo.table_partition_boundary
36 where value = 20211001
37 GO

```

The results table has four columns: name, boundary_id, value, and rows. The data is as follows:

name	boundary_id	value	rows
Transaction_Partitioned	1	20211001	0
Transaction_Partitioned_before_oct	1	20211001	34677

Figure 10.26 – Moving data to the archival table

We can see that all the rows from the first partition of `Transaction_Partitioned` have been now moved to the `Transaction_Partitioned_before_oct` table as the `rows` column indicates a zero for the first partition of `Transaction_Partitioned`.

8. As we had moved partition 1 from `Transaction_Partitioned` and it is empty, it makes sense to remove the partition. This can be done using the `MERGE` command in the partitioned tables. The `MERGE` command works if the partition is empty. Copy and paste the command into the same query window and run the query:

```

ALTER TABLE Transaction_
Partitioned MERGE RANGE('20211001')

```

```
GO  
Select name,boundary_id,value  
from dbo.table_partition_boundary  
where name = 'Transaction_Partitioned'
```

In the `ALTER TABLE` command, we used the `MERGE` command and specified the empty boundary value of `20211001` (October 1) as the input for the `RANGE` clause. This caused the partition with a boundary value of `20211001` (October 1) to be merged with partition 2 (the next partition in line with a boundary value of `20211101`, November 1). To verify, we queried the `table_partition_boundary` view and six partitions. Partition 1's boundary value has changed to `20211101` (November 1), and the partition with a boundary value of `20211001` (October 1) was removed:

```
48  ALTER TABLE Transaction_Partitioned MERGE RANGE('20211001')  
49  GO  
50  Select name,boundary_id,value  
51  from dbo.table_partition_boundary  
52  where name = 'Transaction_Partitioned'  
53  
54
```

Results Messages

View **Table** Chart Export results

Search

name	boundary_id	value
Transaction_Partitioned	1	20211101
Transaction_Partitioned	2	20211201
Transaction_Partitioned	3	20220101
Transaction_Partitioned	4	20220201
Transaction_Partitioned	5	20220301
Transaction_Partitioned	(NULL)	(NULL)

00:00:04 Query executed successfully.

Figure 10.27 – Removing the empty partition

With the preceding steps, we have moved a partition to an archive table and removed the empty partition from the original table.

How it works...

Performing archival operations is extremely challenging in tables involving millions or billions of rows. Archiving the data using traditional `delete` and `insert` statements can be extremely time-consuming, running into hours or even days. It can also block other queries for long periods. However, if you use partition commands such as `SWITCH/MERGE`, as we have done here, the movement of data works seamlessly, that is, in a matter of seconds. This is made possible because behind the scenes, there are no physical data movements for the preceding steps, and the engine remaps the partition to the archival table provided the tables involved adhere to the requirements of the `SWITCH` partition command.

Implementing workload management in an Azure Synapse dedicated SQL pool

How often have we seen a scenario where we have queries coming from a less important system hogging most of the resources and causing fewer resources to be available for business-critical queries, eventually resulting in dissatisfied customers/stakeholders? Wouldn't it be nice to classify the queries based on their business needs and reserve and allocate resources accordingly?

Workload management in a Synapse dedicated SQL pool helps to classify the queries based on the user account used to run the query, the user role, and the application used and map them to resource classes. Resource classes are predefined resource pools with resource limits set as a percentage of the total number of resources. By defining rules that classify the queries and mapping them to resource classes, we could reserve resources based on the business needs and ensure that critical queries get the right amount of resources most of the time.

In this recipe, we will classify the query in a resource class and verify the resource allocated to the query.

Getting ready

Create a Synapse Analytics workspace as explained in the *Provisioning an Azure Synapse Analytics workspace* recipe of *Chapter 8, Processing Data Using Azure Synapse Analytics*.

Complete the *Loading data into dedicated SQL pools using PolyBase using T-SQL* recipe to create a dedicated SQL pool named `packtadesqlpool`, an external table named `dbo.ext_transaction_tbl`, and a dedicated SQL pool table named `dbo.transaction_tbl`. Alternatively, you can use the `Create_External_Table.SQL` script from <https://github.com/PacktPublishing/Azure-Data-Engineering-Cookbook-2nd-edition/tree/main/chapter10> to create the `dbo.ext_transaction_tbl` and `dbo.transaction_tbl` tables.

How to do it...

In this recipe, we will be performing the following tasks:

- Scaling up the data warehouse to **DWU 300** as it's required for this recipe
- Creating a SQL account, running a query, and measuring the resource allocated

- Creating a workload classifier function and mapping the classifier function to the workload resource class.
- Rerunning the query executed in the previous step and verifying the resource allocation

Follow these steps:

1. Log in to `portal.azure.com`, go to **All resources**, and search for `packtadesynapse`. Click on the workspace. Click on **Open Synapse Studio**. Click on the develop button (the notebook-like button) on the left-hand side. Click on the + symbol and select **New SQL script**. Select `packtadesqlpool` in the **Connect to** option. Set the database as **Master**. Refer to *steps 1 and 2* in the *How to do it...* section of the *Loading data into dedicated SQL pools using COPY INTO* recipe for detailed screenshots on how to open a new query window. In the new query window, copy and paste the script provided in the following script to scale up the Synapse instance:

```
ALTER DATABASE packtadesqlpool  
MODIFY (SERVICE_OBJECTIVE = 'DW300c');
```

The following is a screenshot of the execution:

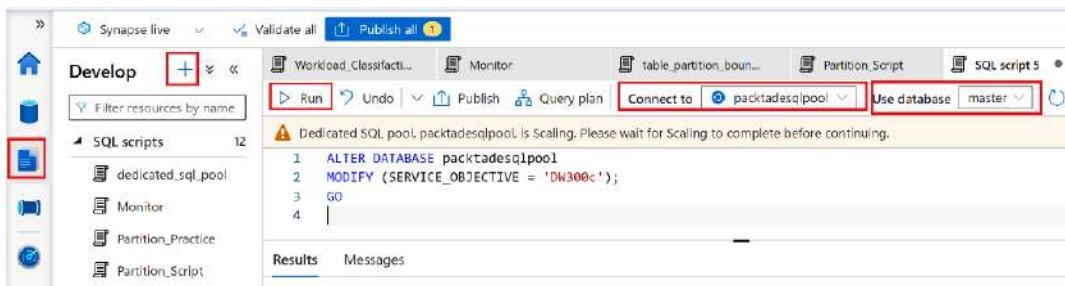


Figure 10.28 – Scaling up the database

As the database needs to scale up, it will pause and resume after a few minutes.

2. Let's create a user account named `AppUser` and run a heavy query using that user account. Connect to the `packtadesqlpool` database. Copy and paste the script into the same query window and run the query:

```
CREATE USER AppUser without login  
GO  
GRANT SELECT ON dbo.transaction_tbl to AppUser  
GO  
EXECUTE AS USER = 'AppUser'  
Select t1.pid,t1.c1,t2.c2,sum(t2.order_count)
```

```

FROM dbo.transaction_tbl t1
inner join dbo.transaction_tbl t2 on t1.transaction_
date = t2.transaction_date
WHERE t1.tid < 1000
Group by t1.pid,t1.c1,t2.c2
order by sum(t2.order_count)
REVERT;
GO

```

The EXECUTE AS USER command switches the context to the AppUser account. The select query takes a few minutes to run. The REVERT command puts the context back into the original account:

The screenshot shows the Azure Data Studio interface with the following details:

- Top Bar:** Workload_Classificati..., Monitor, table_partition_bound..., SQL script 5, Workload_Classificati... (disabled).
- Toolbar:** Cancel, Undo, Publish, Query plan, Connect to (set to packtadesqlpool), Use database (set to packtadesqlpool), Workload_Classificati... (disabled).
- Query Editor:** Contains the following T-SQL script, with the entire script highlighted by a red box:


```

1 ALTER DATABASE packtadesqlpool
2 MODIFY (SERVICE_OBJECTIVE = 'DW300c');
3 GO
4 CREATE USER AppUser without login
5 GO
6 GRANT SELECT ON dbo.transaction_tbl TO AppUser
7 GO
8 EXECUTE AS USER = 'AppUser'
9 Select t1.pid,t1.c1,t2.c2,sum(t2.order_count)
10 FROM dbo.transaction_tbl t1
11 inner join dbo.transaction_tbl t2 on t1.transaction_date = t2.transaction_date
12 WHERE t1.tid < 1000
13 Group by t1.pid,t1.c1,t2.c2
14 order by sum(t2.order_count)
15 REVERT;
16 GO
17
      
```
- Results Grid:** Shows the output of the query:

pid	c1	c2	(No column name)
1	13CB8A65-E8CF-4119-A63B-CBC...	477ED8C5-7A26-47F1-9B5B-D09...	0
1	54D1CCD7-93EF-491A-915E-262...	96120990-3B00-4A1A-9F36-D42...	0
1	A96F8BA3-55BB-4CCC-9D8F-8C0...	96120990-3B00-4A1A-9F36-D42...	0
2	04DD1642-790A-4B3D-B14F-091...	3342F45A-310E-4EEA-8223-B309...	0

Figure 10.29 – Running a query without workload management

3. While the query is running, click on the + button and open a new query window. Execute the script as follows:

```

SELECT req.request_id, classifier_name, group_name,
       command,resource_allocation_percentage
  FROM sys.dm_pdw_exec_requests req
 INNER join sys.dm_pdw_exec_sessions ses on req.session_id = ses.session_id and req.request_id = ses.request_id
 ORDER BY submit_time DESC

```

We can notice that the script was, by default, classified into a resource class called `smallrc` and was granted 8.25 % of the resources:

The screenshot shows the Azure Synapse Studio interface with a query results table. The table has columns: request_id, classifier_name, group_name, command, and resource_allocation_percentage. Two rows are visible: one with request_id QID6499 and another with request_id QID6491. The row for QID6491 is highlighted with a red box around its entire row. The 'group_name' column for QID6491 contains 'smallrc'. The 'resource_allocation_percentage' column for QID6491 contains '8.25'.

request_id	classifier_name	group_name	command	resource_allocation_percentage
QID6499	(NULL)	(NULL)	SELECT req.request_id, classifier_name, grou...	(NULL)
QID6491	(NULL)	smallrc	Select t1.pid,t1.c1,t2.c2,sum(t2.order_count) ...	8.25

Figure 10.30 – Checking the number of resources consumed

4. Let's create a classifier function and rerun the same expensive query. Copy and paste the following script into the same query window and run it:

```

CREATE WORKLOAD CLASSIFIER WC_AppUser WITH
( WORKLOAD_GROUP = 'mediumrc'
, MEMBERNAME = 'AppUser'
)
GO
EXECUTE AS USER = 'AppUser'

```

```

Select t1.pid,t1.c1,t2.c2,sum(t2.order_count)
FROM dbo.transaction_tbl t1
inner join dbo.transaction_tbl t2 on t1.transaction_
date = t2.transaction_date
WHERE t1.tid < 1000
Group by t1.pid,t1.c1,t2.c2
order by sum(t2.order_count)
REVERT;
GO

```

We have created a workload classifier function named WC_AppUser. In the CREATE WORKLOAD CLASSIFIER function, we specify the workload group as mediumrc. Mediumrc is a built-in resource class that provides 10% of resource allocation for queries in databases at the DWU 300 service level:

```

16 GO
17 CREATE WORKLOAD CLASSIFIER WC_AppUser WITH
18 ( WORKLOAD_GROUP = 'mediumrc'
19 , MEMBERNAME = 'AppUser'
20 )
21 GO
22 EXECUTE AS USER = 'AppUser'
23 Select t1.pid,t1.c1,t2.c2,sum(t2.order_count)
24 FROM dbo.transaction_tbl t1
25 inner join dbo.transaction_tbl t2 on t1.transaction_date = t2.transaction_date
26 WHERE t1.tid < 1000
27 Group by t1.pid,t1.c1,t2.c2
28 order by sum(t2.order_count)
29 REVERT;
30 GO

```

Results **Messages**

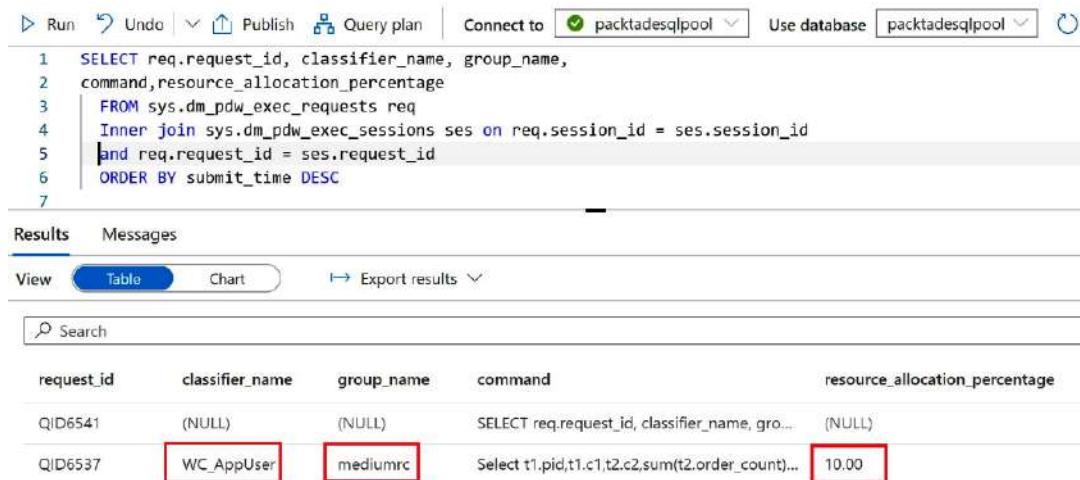
View **Table** **Chart** **Export results**

Search

pid	c1	c2	(No column name)
1	13CB8A65-E8CF-4119-A63B-CBC...	477ED8C5-7A26-47F1-9B5B-D09...	0
1	54D1CCD7-93EF-491A-915E-262...	96120990-3B00-4A1A-9F36-D42...	0
1	A96F8BA3-55BB-4CCC-9D8F-8C0...	96120990-3B00-4A1A-9F36-D42...	0
2	04DD1642-790A-4B3D-B14F-091...	3342F45A-310E-4EEA-8223-B309...	0

Figure 10.31 – Running a query with the classifier function

5. Click on the + button, select **SQL Script**, and open a new query window. Then, execute the same script used in step 3 to monitor the resource allocation:



The screenshot shows a query editor interface with the following details:

- Toolbar: Run, Undo, Publish, Query plan, Connect to (selected), Use database (packtadesqlpool), and a refresh icon.
- Query Text (lines 1-7):

```
1 SELECT req.request_id, classifier_name, group_name,
2 command,resource_allocation_percentage
3     FROM sys.dm_pdw_exec_requests req
4     Inner join sys.dm_pdw_exec_sessions ses on req.session_id = ses.session_id
5     |and req.request_id = ses.request_id
6     | ORDER BY submit_time DESC
7
```
- Results Tab: Shows a table with the following data:

request_id	classifier_name	group_name	command	resource_allocation_percentage
QID6541	(NULL)	(NULL)	SELECT req.request_id, classifier_name, gro...	(NULL)
QID6537	WC_AppUser	mediumrc	Select t1.pid,t1.c1;t2.c2,sum(t2.order_count)...	10.00

Figure 10.32 – Checking the resource allocation

We can see that the query has been classified using the `WC_AppUser` classifier function and has been allocated 10% of resources using the `mediumrc` resource class.

How it works...

A Synapse dedicated SQL pool offers four dynamic resource classes, namely `smallrc`, `mediumrc`, `largerc`, and `xlargerc`. The `smallrc`, `mediumrc`, `largerc`, and `xlargerc` resource classes offer 8%, 10%, 22%, and 70% of resource usage, respectively, at DWU 300C. The allocation percentages for these resource classes vary marginally depending on the DWU. Detailed information on the percentage allocation of resources is provided at <https://docs.microsoft.com/en-us/azure/synapse-analytics/sql-data-warehouse/resource-classes-for-workload-management>.

Mapping the query to a resource class by classifying it using a user account helps us to set resource limits on workloads from different user accounts. You could map user accounts to higher resource classes such as `largerc` and `xlargerc` for more critical and resource-intensive queries.

Creating workload groups for advanced workload management

Workload groups in Synapse dedicated pools allow you to define resource pools with custom resource allocation percentages compared to the predefined percentages offered by resource classes. Additionally, workload groups offer the flexibility to set minimum and maximum resource percentages for the entire pool and also for each request. Setting a minimum resource percentage for a workload group ensures there will always be a percentage of resource reserved for the queries mapped to the resource group. In this recipe, we will define a custom resource group with minimum and maximum resource percentages for the pool and for each request, and we will also classify it in such a way that the resource allocation changes depending on the time of query execution.

Getting ready

Create a Synapse Analytics workspace as explained in the *Provisioning an Azure Synapse Analytics workspace* recipe of *Chapter 8, Processing Data Using Azure Synapse Analytics*.

Complete the *Loading data into dedicated SQL pools using PolyBase using T-SQL* recipe to create a dedicated SQL pool named packtadesqlpool, an external table named dbo.ext_transaction_tbl, and a dedicated SQL pool table named dbo.transaction_tbl. Alternatively, you can use the Create_External_Table.SQL script from <https://github.com/PacktPublishing/Azure-Data-Engineering-Cookbook-2nd-edition/tree/main/chapter10> to create the dbo.ext_transaction_tbl and dbo.transaction_tbl tables.

Complete the *Implementing workload management in an Azure Synapse dedicated SQL pool* recipe to create a workload classifier function and map it to the workload resource class.

How to do it...

In this recipe, we will be performing the following tasks:

- Creating a workload group with minimum and maximum resource percentages for the pool and per request.
- Creating a classifier function based on user account and login time.
- Testing queries at different times to see the resource allocation.

Detailed steps are provided as follows:

1. Log in to portal.azure.com, go to **All resources**, and search for packtadesynapse. Click on the workspace. Click on **Open Synapse Studio**. Click on the develop button (the notebook-like button) on the left-hand side. Click on the + symbol and select **New SQL script**. Select packtadesqlpool in the **Connect to** option. Copy and paste the following script and then hit the **Run** button:

```
CREATE WORKLOAD GROUP WG_AppUser_offpeak WITH
( REQUEST_MIN_RESOURCE_GRANT_PERCENT = 12
,REQUEST_MAX_RESOURCE_GRANT_PERCENT = 20
,MIN_PERCENTAGE_RESOURCE = 24
,CAP_PERCENTAGE_RESOURCE = 40
)
GO
CREATE WORKLOAD CLASSIFIER WC_AppUser_offpeak WITH
( WORKLOAD_GROUP = 'WG_AppUser_offpeak'
, MEMBERNAME = 'AppUser'
, START_TIME = '13:30'
, END_TIME = '23:00'
)
```

The preceding script creates a workload group called `WG_AppUser_offpeak`. The `WG_AppUser_offpeak` configuration is explained as follows:

- `REQUEST_MIN_RESOURCE_GRANT_PERCENT` of 12% implies any query classified to map to `WG_AppUser_offpeak` will get a minimum of 12% of total resources in the database.
- `REQUEST_MAX_RESOURCE_GRANT_PERCENT` of 20% implies a query classified to map to `WG_AppUser_offpeak` can get a maximum of 20% of total resources in the database.
- `MIN_PERCENTAGE_RESOURCE` of 24% implies 24% of total resources in the database will be reserved for the `WG_AppUser_offpeak` pool.
- `CAP_PERCENTAGE_RESOURCE` of 40% indicates the total resource usage of all the queries in `WG_AppUser_offpeak` can reach up to 40%.

Additionally, the script creates a classifier function named `WC_AppUser_offpeak`, which maps the `AppUser` account with the `WG_AppUser_offpeak` workload group. The classifier needs to map the queries from the `AppUser` account to `WC_AppUser_offpeak` only during off-peak hours between 9:30 p.m. and 7 a.m. Singapore time. The `START_TIME` and `END_TIME` values are set in UTC time (8 hours behind Singapore time) and, hence, are specified as 13:30 and 23:00, respectively:

The screenshot shows the Microsoft SQL Server Management Studio (SSMS) interface. The top menu bar includes tabs for 'SQL script 6', 'SQL script 9', 'SQL script 11', and a dropdown for 'S'. Below the menu are buttons for 'Run', 'Undo', 'Publish', 'Query plan', and 'Connect to' (set to 'packtadesqlpool'). The main area contains a numbered T-SQL script:

```

1 CREATE WORKLOAD GROUP WG_AppUser_offpeak WITH
2   ( REQUEST_MIN_RESOURCE_GRANT_PERCENT = 12
3     ,REQUEST_MAX_RESOURCE_GRANT_PERCENT = 20
4     ,MIN_PERCENTAGE_RESOURCE = 24
5     ,CAP_PERCENTAGE_RESOURCE = 40
6   )
7 GO
8 CREATE WORKLOAD CLASSIFIER WC_AppUser_offpeak WITH
9   ( WORKLOAD_GROUP = 'WG_AppUser_offpeak'
10    ,MEMBERNAME = 'AppUser'
11    ,START_TIME = '13:30'
12    ,END_TIME = '23:00'
13  )
14

```

Specific parts of the script are highlighted with red boxes: the resource grant percentages (12, 20, 24, 40), the start time ('13:30'), and the end time ('23:00'). At the bottom of the interface, there are tabs for 'Results' and 'Messages', with 'Results' being the active tab.

Figure 10.33 – Configuring the workload management

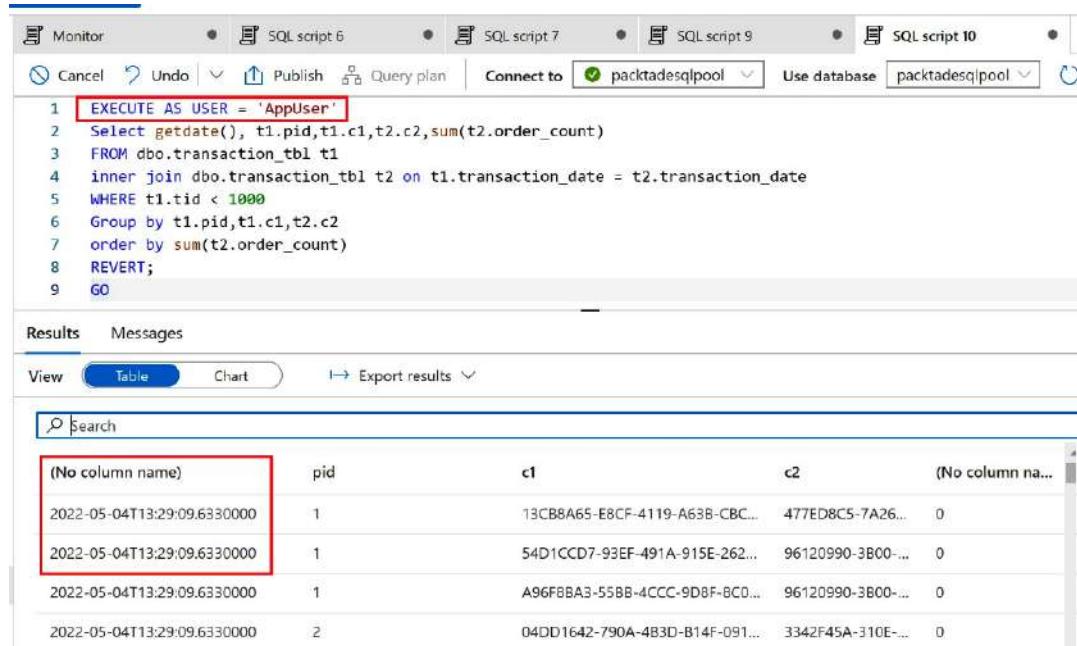
- Let's run the expensive query using AppUser before the off-peak time, which starts at 13:30 hours UTC. Copy the following script and run it in a new query window before the start time of the off-peak period:

```

EXECUTE AS USER = 'AppUser'
Select getdate(), t1.pid,t1.c1,t2.c2,sum(t2.order_count)
FROM dbo.transaction_tbl t1
inner join dbo.transaction_tbl t2 on t1.transaction_
date = t2.transaction_date
WHERE t1.tid < 1000
Group by t1.pid,t1.c1,t2.c2
order by sum(t2.order_count)
REVERT;
GO

```

The output of the script is shown in the following screenshot:



```
1 EXECUTE AS USER = 'AppUser'
2 Select getdate(), t1.pid,t1.c1,t2.c2,sum(t2.order_count)
3 FROM dbo.transaction_tbl t1
4 inner join dbo.transaction_tbl t2 on t1.transaction_date = t2.transaction_date
5 WHERE t1.tid < 1000
6 Group by t1.pid,t1.c1,t2.c2
7 order by sum(t2.order_count)
8 REVERT;
9 GO
```

Results Messages

View Table Chart Export results

Search

(No column name)	pid	c1	c2	(No column na...)
2022-05-04T13:29:09.6330000	1	13CB8A65-E8CF-4119-A63B-CBC...	477ED8C5-7A26...	0
2022-05-04T13:29:09.6330000	1	54D1CCD7-93EF-491A-915E-262...	96120990-3B00...	0
2022-05-04T13:29:09.6330000	1	A96F8BA3-55BB-4CCC-9D8F-8C0...	96120990-3B00...	0
2022-05-04T13:29:09.6330000	2	04DD1642-790A-4B3D-B14F-091...	3342F45A-310E...	0

Figure 10.34 – Running a query before the off-peak period

The first column returns the current time in UTC, which is 13:29, just before the start of the off-peak period.

3. While the query is running, click on the + button, select **SQL Script**, and open a new query window. Run the following script to check the resource class and the resources allocated for the query:

```
SELECT req.request_id, classifier_name, resource_class,
       command,resource_allocation_percentage,submit_time
  FROM sys.dm_pdw_exec_requests req
 INNER join sys.dm_pdw_exec_sessions ses on req.session_id = ses.session_id
    and req.request_id = ses.request_id
 ORDER BY submit_time DESC
```

The output of the script is shown in the following screenshot:

The screenshot shows the Azure Synapse Studio interface. At the top, there are tabs for 'Monitor', 'SQL script 6', 'SQL script 7', 'SQL script 9', and 'SQL script 10'. Below these are buttons for 'Run', 'Undo', 'Publish', 'Query plan', 'Connect to', 'Use database', and a refresh icon. The 'Connect to' dropdown is set to 'packtadesqlpool' and 'Use database' is set to 'packtadesolpool'. The main area contains a query script:

```

1  SELECT req.request_id, classifier_name, resource_class,
2  command,resource_allocation_percentage,submit_time
3  FROM sys.dm_pdw_exec_requests req
4  Inner join sys.dm_pdw_exec_sessions ses on req.session_id = ses.session_id
5  and req.request_id = ses.request_id
6  ORDER BY submit_time DESC

```

Below the script, there are tabs for 'Results' and 'Messages', with 'Results' being the active tab. Under 'Results', there are buttons for 'View' (set to 'Table'), 'Chart', and 'Export results'. A search bar is also present. The results table has columns: request_id, classifier_name, resource_class, command, resource_allocation_percentage, and submit_time. Two rows are shown:

request_id	classifier_name	resource_class	command	resource_allocation_percentage	submit_time
QID6994	(NULL)	(NULL)	SELECT req.request_id, classifier_name, resour...	(NULL)	2022-05-04T13:29:15.6400000
QID6990	WC_AppUser	mediumrcc	Select getdate(), t1.pid,t1.c1,t2.c2,sum(t2.order...	10.00	2022-05-04T13:29:09.6030000

Figure 10.35 – Monitoring the query before the off-peak period

The preceding script execution shows that the script has been classified using the `WC_AppUser` classifier function (which was created in the *Implementing workload management in an Azure Synapse dedicated SQL pool* recipe) and has been allocated 10% of resources via the `mediumrcc` resource class. The `submit_time` column shows the time as 13:29, which is just before the off-peak period that starts at 13:30 hours.

4. Let's run the select query again using the `AppUser` account but this time inside the off-peak period. Copy the script from *step 2* and rerun it. The `Getdate()` function returns 13:44, which is in the off-peak period defined:

The screenshot shows a SQL Server Management Studio (SSMS) interface. At the top, there are tabs for 'SQL script 6' through 'SQL script 12'. Below the tabs, the 'Connect to' dropdown is set to 'packtadesqlpool' and the 'Use database' dropdown is set to 'packtadesqlpool'. The main area contains a query window with the following T-SQL script:

```

1 EXECUTE AS USER = 'AppUser'
2 Select getdate(), t1.pid,t1.c1,t2.c2,sum(t2.order_count)
3 FROM dbo.transaction_tbl t1
4 inner join dbo.transaction_tbl t2 on t1.transaction_date = t2.transaction_date
5 WHERE t1.tid < 1000
6 Group by t1.pid,t1.c1,t2.c2
7 order by sum(t2.order_count)
8 REVERT;
9 GO

```

Below the script, the 'Results' tab is selected, showing the output in 'Table' view. The results are as follows:

(No column name)	pid	c1	c2	(No column na...)
2022-05-04T13:44:24.8330000	1	13CB8A65-E8CF-4119-A63B-CBC...	477ED8C5-7A26...	0
2022-05-04T13:44:24.8330000	1	54D1CCD7-93EF-491A-915E-262...	96120990-3B00...	0
2022-05-04T13:44:24.8330000	1	A96F8BA3-55BB-4CCC-9D8F-8C0...	96120990-3B00...	0
2022-05-04T13:44:24.8330000	2	04DD1642-790A-4B3D-B14F-091...	3342F45A-310E...	0

Figure 10.36 – Running a query during the off-peak period

- Click on the + button, select **SQL Script**, and open a query window. Run the monitoring script used in *step 3* to check the resource allocation:

The screenshot shows a SQL Server Management Studio (SSMS) interface. At the top, there are tabs for 'SQL script 6' through 'SQL script 12'. Below the tabs, the 'Connect to' dropdown is set to 'packtadesqlpool' and the 'Use database' dropdown is set to 'packtadesqlpool'. The main area contains a query window with the following T-SQL script:

```

1 SELECT req.request_id, classifier_name, resource_class,
2 command,resource_allocation_percentage,submit_time
3 FROM sys.dm_pdw_exec_requests req
4 Inner join sys.dm_pdw_exec_sessions ses on req.session_id = ses.session_id
5 and req.request_id = ses.request_id
6 ORDER BY submit_time DESC

```

Below the script, the 'Results' tab is selected, showing the output in 'Table' view. The results are as follows:

request_id	classifier_name	resource_class	command	resource_allocation_percentage	submit_time
QID7090	(NULL)	(NULL)	SELECT req.request_id, classifier_name, res...	(NULL)	2022-05-04T13:44:50.0530000
QID7082	WC_AppUser_offpeak	WG_AppUser_offpeak	Select getdate(), t1.pid,t1.c1,t2.c2,sum(t2.or...	20.00	2022-05-04T13:44:24.8030000

Figure 10.37 – Running a query during the off-peak period

We noticed that as the `submit_time` query was in the off-peak period, it was mapped to the `WG_AppUser_offpeak` workload group. The query is allocated 20% of resources, as the maximum resource per query of the `WG_AppUser_offpeak` workload group is 20%.

How it works...

Using workload groups, we have more flexibility in allocating resources per query level and for the entire resource pool, too. In this example, there were two classifier functions, namely `WG_AppUser_offpeak` and `WG_AppUser`, that were mapped to the `AppUser` account. `WG_AppUser_offpeak` had an additional mapping based on query submission time; therefore, when the classifier function matched both the user account and query submission time, the `WG_AppUser_offpeak` classifier was selected and appropriate resources were allocated.

It is important to note that allocating more resources per query is good for getting heavy tasks done but would limit the number of users accessing the resource pool concurrently. It is essential to strike a balance between concurrency and resources, and workload groups help us to strike that balance. In the preceding example, the application needs to support more users during peak hour operations; therefore, we allocated 10% of resources for the query via the `WG_AppUser` workload group. However, maintenance jobs during off-peak hours require additional resources, so 20% of resources were allocated via `WG_AppUser_offpeak`. We had set a cap on total resources in the `WG_AppUser_offpeak` workload group to 40%, which implies it had the capacity to support just another query with 20% resource allocation. However, that still works, as during off-peak hours, the resource pool is not expected to support many concurrent users.

11

Monitoring Synapse SQL and Spark Pools

As introduced in *Chapter 8, Processing Data Using Azure Synapse Analytics*, Azure Synapse Analytics is comprised of three key components – a Synapse integration pipeline to ingest and transform the data, SQL pool and Spark pool to process and serve the data, and Power BI integration to visualize the data. Monitoring the health of Synapse SQL and Spark pools is an integral part of the work of a data engineer managing large data engineering projects.

In this chapter, you will learn how to integrate Synapse SQL and Spark pools with Azure Log Analytics, identify long-running Spark jobs and SQL queries using a Log Analytics workspace, monitor Synapse health from Azure Monitor, and check the table and index health of SQL dedicated pool tables using **Dynamic Management Views (DMVs)**.

In this chapter, we will cover the following recipes:

- Configuring a Log Analytics workspace for Synapse SQL pools
- Configuring a Log Analytics workspace for Synapse Spark pools
- Using Kusto queries to monitor SQL and Spark pools
- Creating workbooks in a Log Analytics workspace to visualize monitoring data
- Monitoring table distribution, data skew, and index health using Synapse DMVs
- Building monitoring dashboards for Synapse with Azure Monitor

Technical requirements

For this chapter, you will need a Microsoft Azure subscription.

Configuring a Log Analytics workspace for Synapse SQL pools

A Log Analytics workspace is an Azure service that's used to store the diagnostic logs of several Azure services in a single place. With a Log Analytics workspace, you can store the performance metric data of a Synapse SQL pool, such as how much DWU was used, I/O usage, queued queries, query-level consumption details, and more. In this recipe, we will configure a Log Analytics workspace for a Synapse SQL pool.

Getting ready

To get started, log in to <https://portal.azure.com> using your Azure credentials:

- Create a Synapse Analytics workspace, as explained in the *Provisioning an Azure Synapse Analytics workspace* recipe of *Chapter 8, Processing Data Using Azure Synapse Analytics*.
- Create a dedicated Synapse SQL pool database, as explained in *steps 1 to 3* in the *How to do it...* section of the *Loading data into a dedicated SQL pool using PolyBase and T-SQL* recipe of *Chapter 10, Building the Serving Layer in an Azure Synapse SQL Pool*.

How to do it...

First, let's create a Log Analytics workspace:

1. Go to the portal.azure.com home page and click on **Create a resource**. Search for **Log Analytics** and select **Log Analytics Workspace**. Click on the **Create** button:

Home > Create a resource >

Log Analytics Workspace

Microsoft



Figure 11.1 – Creating a Log Analytics workspace

2. Set the resource group name as `PacktADESynapse`. Set the instance name as `PacktADELogAnalytics`. Pick the same location as that of your Synapse workspace. Then, click on **Review + Create**:

Home > Create a resource > Log Analytics Workspace >

Create Log Analytics workspace

Basics Tags Review + Create

i A Log Analytics workspace is the basic management unit of Azure Monitor Logs. There are specific considerations you should take when creating a new Log Analytics workspace. [Learn more](#) X

With Azure Monitor Logs you can easily store, retain, and query data collected from your monitored resources in Azure and other environments for valuable insights. A Log Analytics workspace is the logical storage unit where your log data is collected and stored.

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 * ⓘ

PacktADESynapse

Create new

Instance details

Name * ⓘ

PacktADELogAnalytics

Region * ⓘ

East US

Review + Create

« Previous

Next : Tags >

Figure 11.2 – Providing a Log Analytics workspace name

3. Log in to `portal.azure.com`, go to **All resources**, and search for `packtadesqlpool`, which is the dedicated Synapse SQL pool you created in the *Loading data into a dedicated SQL pool using PolyBase and T-SQL* recipe of Chapter 10, *Building the Serving Layer in Azure Synapse SQL Pool*. Click on **SQL pool** and search for **Diagnostics Settings**:

The screenshot shows the Azure portal interface for a Dedicated SQL pool named 'packtadesqlpool'. The left sidebar has a 'Monitoring' section with a 'Diagnostic settings' link, which is highlighted with a red box. The main content area displays a message: 'This dedicated SQL pool is currently paused. You may experience limited to no connectivity.' Below this, there's an 'Essentials' section showing the 'Resource group (move)' as 'PacktADESynapse' and the 'Workspace name' as 'PacktADESynapse'. There are also 'Resume', 'Scale', 'Restore', 'New restore point', 'Delete', and 'Open in Synapse Studio' buttons at the top.

Figure 11.3 – Open dedicated SQL pool diagnostics

4. Click on + Add diagnostic setting:

The screenshot shows the 'Diagnostic settings' page for the same Dedicated SQL pool. The left sidebar has a 'Diagnostic settings' link, which is highlighted with a red box. The main content area shows a table titled 'Diagnostic settings' with columns: 'Name', 'Storage account', 'Event hub', and 'Log Analy'. A single row is listed with the message 'No diagnostic settings defined'. At the bottom of the table, there is a blue button labeled '+ Add diagnostic setting', which is also highlighted with a red box.

Figure 11.4 – Adding diagnostics

5. Set **Diagnostic setting name** to `Synapsesdedicatedpooldiag`. Then, select all the options under **Categories**. Click on the **Send to Log Analytics Workspace** checkbox and select `PacktADELogAnalytics` for **Log Analytics workspace**. Then, click **Save**:

Home > All resources > packtadesqlpool (packtadesynapse/packtadesqlpool) >

Diagnostic setting ...

Save Discard Delete Feedback

A diagnostic setting specifies a list of categories of platform logs and/or metrics that you want to collect from a resource, and one or more destinations that you would stream them to. Normal usage charges for the destination will occur. [Learn more about the different log categories and contents of those logs](#)

Diagnostic setting name *

Logs

Categories

- SqlRequests
- RequestSteps
- ExecRequests
- DmsWorkers
- Waits

Destination details

Send to Log Analytics workspace

Subscription

Log Analytics workspace

Archive to a storage account

Stream to an event hub

Send to partner solution

Figure 11.5 – Adding diagnostics

How it works...

As a first step, we created a Log Analytics workspace and then we linked the diagnostic setting in the dedicated SQL pool to it. As the queries are executed in the dedicated SQL pool, performance metrics data about the queries will be recorded in the Log Analytics workspace. The granularity/depth of the performance metric data recorded depends on the log categories selected while configuring the diagnostic settings of the dedicated SQL pool. We selected all five categories of logs available to gather detailed information about the SQL pool's performance. A brief description of the log categories is as follows:

- **SqlRequests:** Provides information about query steps at the node or distribution level for SQL requests
- **RequestSteps:** Provides step-level information for each recorded query
- **ExecRequests:** Provides information about all the queries that are executed in the SQL pool

- **DmsWorkers:** Provides information about the worker process moving data across nodes
- **Waits:** Provides information about various wait categories (**CPU/Memory/Locks/IO/Network**) that queries were spending time in

Accessing the data in a Log Analytics workspace and finding key performance insights will be covered in the subsequent recipes of this chapter.

Configuring a Log Analytics workspace for Synapse Spark pools

In this recipe, we will explore integrating a Log Analytics workspace with Synapse Spark pools.

Getting ready

To get started, log in to <https://portal.azure.com> using your Azure credentials:

- Create a Synapse Analytics workspace, as explained in the *Provisioning an Azure Synapse Analytics workspace* recipe of *Chapter 8, Processing Data Using Azure Synapse Analytics*.
- Create a Spark pool cluster, as explained in the *Provisioning and configuring Spark pools* recipe of *Chapter 8, Processing Data Using Azure Synapse Analytics*.
- Create a Log Analytics workspace (if you haven't already done so), as explained in steps 1 and 2 in the *How to do it...* section of the *Configuring a Log Analytics Workspace for Synapse dedicated SQL Pools* recipe in this chapter.

How to do it...

Follow these steps to integrate Synapse Spark pools with a Log Analytics workspace:

1. Download the `spark_loganalytics_conf.txt` file from <https://github.com/PacktPublishing/Azure-Data-Engineering-Cookbook-2nd-edition/blob/main/chapter11/>.
2. Log in to `portal.azure.com`, go to **All resources**, and search for **PacktADELogAnalytics**, which is the Log Analytics workspace you created in the *Configuring a Log Analytics workspace for Synapse dedicated SQL pools* recipe of this chapter. Click on **Agents Management** and copy the **Workspace ID** and **Primary key** information on the right:

The screenshot shows the 'Agents management' page of a Log Analytics workspace. On the left, there's a sidebar with various settings like Overview, Activity log, and Tags. The 'Agents management' option is selected and highlighted with a red box. The main area shows '0 Windows computers connected' and provides links to download the Windows Agent (64-bit or 32-bit). Below that, the 'Workspace ID' and 'Primary key' are displayed in input fields, both of which are also highlighted with red boxes. There are 'Regenerate' buttons next to each key.

Figure 11.6 – Workspace details

3. Open the `spark_loganalytics_conf.txt` file you downloaded in step 1. Paste the workspace ID after `spark.synapse.logAnalytics.workspaceId` (the second line) in the file. Paste the primary key after `spark.synapse.logAnalytics.secret` (the third line) in the file. Then, save the file:

```
spark.synapse.logAnalytics.enabled true
spark.synapse.logAnalytics.workspaceId 97216307-5c7b-4a76-8df2-fd625d8e03ea
spark.synapse.logAnalytics.secret JIdco9Jqhb3ojI1vjb4re7Ui0SCi0smqKG17h/2w/ptIKNLDw0bE/HaKP...
```

Figure 11.7 – Adding workspace details to the file

4. Log in to `portal.azure.com`, go to **All resources**, and search for `packtsparkpool`, which is the Spark pool you created in the *Provisioning and configuring Spark pools* recipe of *Chapter 8, Processing Data Using Azure Synapse Analytics*. Click on **Spark configuration** under **Settings**. Then, click on **Upload spark config file**:

Home > PacktADESynapse > packtsparkpool (packtadesynapse/packtsparkpool)

The screenshot shows the Apache Spark pool configuration interface. At the top, there's a search bar with 'conf' typed in, a 'Upload spark config file' button with a red box around it, and a 'Refresh' button. Below the search bar are two tabs: 'Access control (IAM)' and 'Spark configuration', with 'Spark configuration' being the active tab and also having a red box around it. A section titled 'User-provisioned spark config' follows, containing a table with one row. The table has columns for 'Name' and 'Size'. The row displays the message: 'No user-provided config file currently uploaded. You can upload "spark config file".'

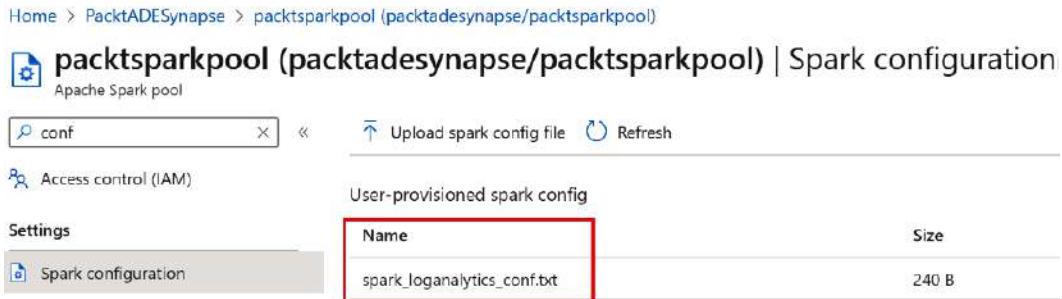
Figure 11.8 – Configuring a Spark pool

- Click on the folder-like icon and select the `spark_loganalytics_conf.txt` file you saved in step 3. Click on the **Upload** button:

The screenshot shows the 'Upload spark config file' dialog box. It includes a close button ('X'), the URL 'packtadesynapse/packtsparkpool', and instructions: 'Upload a Spark configuration file to specify additional properties on the Spark pool. This will be referenced to configure Spark applications upon job submission.' Below this is a 'File upload' section with a red box around it, showing the file path '"spark_loganalytics_conf.txt"' and a blue folder icon with a red box around it. At the bottom are two buttons: 'Force new settings' (radio button) and 'Immediately apply settings change and cancel all active applications.' (checkbox). The 'Upload' button at the very bottom is also highlighted with a red box.

Figure 11.9 – The Upload spark config file screen

6. Once uploaded, the configuration file's name will be reflected on the Spark pool configuration:



The screenshot shows the Azure portal interface for managing a Synapse Spark pool named "packtsparkpool". The URL in the address bar is "Home > PacktADESynapse > packtsparkpool (packtadesynapse/packtsparkpool)". The main title is "packtsparkpool (packtadesynapse/packtsparkpool) | Spark configuration". On the left, there is a sidebar with "Access control (IAM)" and "Settings" sections, and a "Spark configuration" section which is currently selected and highlighted in grey. At the top, there is a search bar with "conf" and a "Upload spark config file" button. Below the search bar, the heading "User-provisioned spark config" is followed by a table. The table has two columns: "Name" and "Size". A single row is listed: "spark_loganalytics_conf.txt" with a size of "240 B". The "Name" column for this row is highlighted with a red rectangular box.

Figure 11.10 – Updated Spark pool

How it works...

Log Analytics for Synapse Spark pools was enabled via the `spark_loganalytics_conf.txt` file's first line – that is, `spark.synapse.logAnalytics.enabled true`. Using the credentials of the Log Analytics workspace specified in `spark_loganalytics_conf.txt`, the Spark pool will log all the performance metrics of all jobs and notebooks executed in the Spark pool in the Log Analytics workspace. Analyzing the performance metric data of a Spark pool and identifying key insights will be covered in the next recipe.

Using Kusto queries to monitor SQL and Spark pools

In the previous two recipes (*Configuring a Log Analytics workspace for Synapse dedicated SQL pools* and *Configuring a Log Analytics workspace for Synapse Spark pools*), we created a Log Analytics workspace and configured the diagnostics logs of Synapse-dedicated SQL and Spark pools to be written to it. In this recipe, we will use Kusto queries to query a Log Analytics workspace to gain insights into the diagnostic logs collected.

Getting ready

To get started, log in to <https://portal.azure.com> using your Azure credentials:

- Complete the *Configuring a Log Analytics workspace for Synapse dedicated SQL Pools* and *Configuring a Log Analytics workspace for Synapse Spark pools* recipes covered earlier in this chapter to configure a Log Analytics workspace for Synapse SQL and Spark pool.

- Complete the *Loading data into a dedicated SQL pool using PolyBase and T-SQL* recipe of *Chapter 10, Building the Serving Layer in Azure Synapse SQL Pool*, to create a dedicated SQL pool named packtadesqlpool, an external table named dbo.ext_transaction_tbl, and a dedicated SQL pool table named dbo.transaction_tbl. Alternatively, to create the dbo.ext_transaction_tbl and dbo.transaction_tbl tables, you may use the `Create_External_Table.sql` script available in <https://github.com/PacktPublishing/Azure-Data-Engineering-Cookbook-2nd-edition/tree/main/chapter10>.

How to do it...

In this recipe, we will do the following:

- Run Kusto queries in a Log Analytics workspace to find long-running SQL queries in a dedicated SQL pool at a particular time.
- Run Kusto queries in a Log Analytics workspace to find high CPU-consuming Spark jobs.

Perform the following steps to find long-running SQL queries in the dedicated SQL pool:

1. The first step is to create a workload on a Synapse SQL pool:
 - I. Download the `SQLPool_Questions.sql` script from https://github.com/PacktPublishing/Azure-Data-Engineering-Cookbook-2nd-edition/blob/main/chapter11/SQLPool_Questions.sql.
 - II. Log in to `portal.azure.com`, go to **All resources**, and search for **packtadesynapse**. Click on the workspace and click on **Open Synapse Studio**. Then, click the develop button (the notebook-like button) on the left. Click on the + symbol and select **Import**:

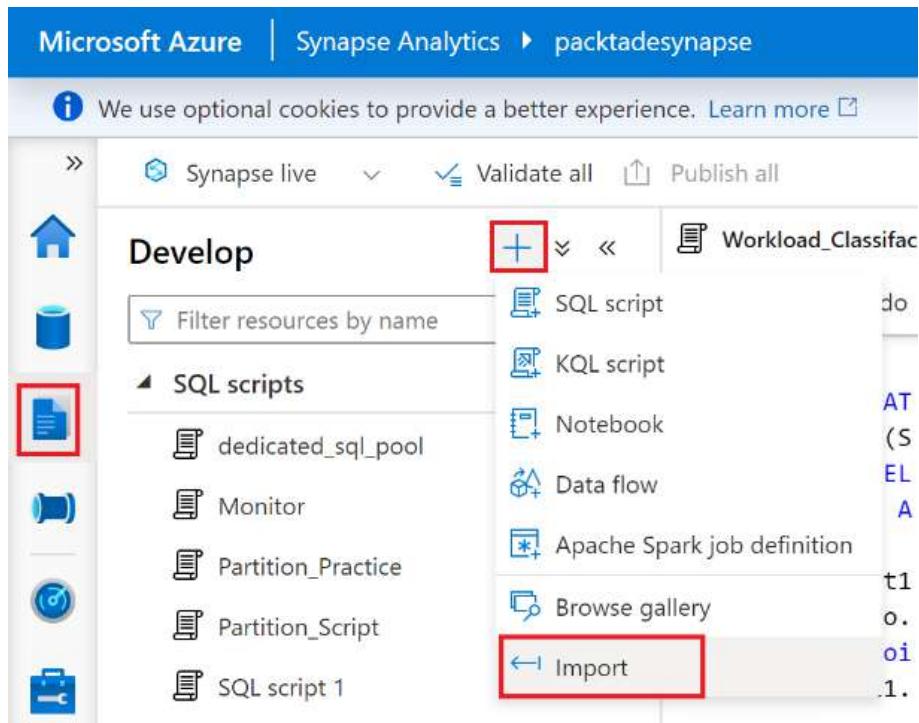
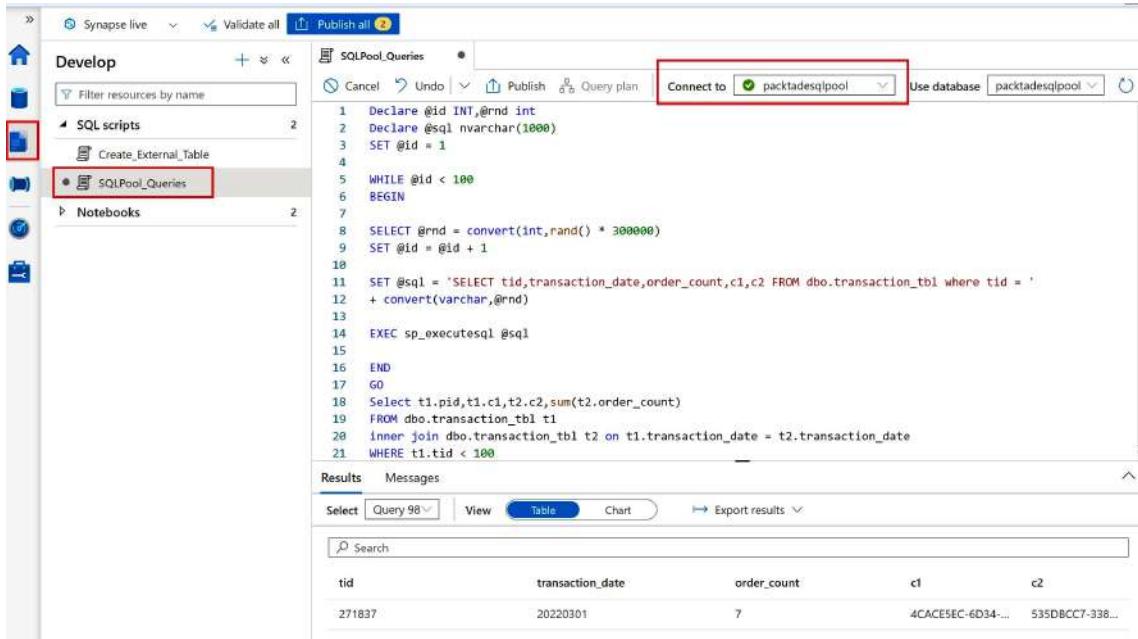


Figure 11.11 – Importing the SQL script

- III. Select the `SQLPool_Queries.sql` file you downloaded in *step I*. Connect to the **packtadesqlpool** database and run the script. The script will return 100 queries in 1 to 2 minutes:



The screenshot shows the Azure Synapse Studio interface. On the left, there's a navigation sidebar with 'Develop' selected. Under 'SQL scripts', a file named 'SQLPool_Qualities.sql' is highlighted with a red box. The main area contains the SQL script content:

```

1 Declare @id INT,@rnd int
2 Declare @sql nvarchar(1000)
3 SET @id = 1
4
5 WHILE @id < 100
6 BEGIN
7
8     SELECT @rnd = convert(int,rand() * 300000)
9     SET @id = @id + 1
10
11    SET @sql = 'SELECT tid,transaction_date,order_count,c1,c2 FROM dbo.transaction_tbl where tid = '
12    + convert(varchar,@rnd)
13
14    EXEC sp_executesql @sql
15
16 END
17 GO
18 Select t1.pid,t1.c1,t2.c2,sum(t2.order_count)
19 FROM dbo.transaction_tbl t1
20 inner join dbo.transaction_tbl t2 on t1.transaction_date = t2.transaction_date
21 WHERE t1.tid < 100

```

Below the script, there are tabs for 'Results' and 'Messages'. The 'Results' tab is selected, showing a table with the following data:

tid	transaction_date	order_count	c1	c2
271837	20220301	7	4CACE5EC-6D34-...	535DBCC7-338...

Figure 11.12 – Running the SQL workload

2. After the `SQLPool_Qualities.sql` file finishes executed, wait 10 to 15 minutes and then go to `portal.azure.com`, go to **All resources**, and search for **PacktADELogAnalytics**, the Log Analytics workspace, and click on it. Click on **Logs**. Close the **Queries** popup using the **close** button at the top right:

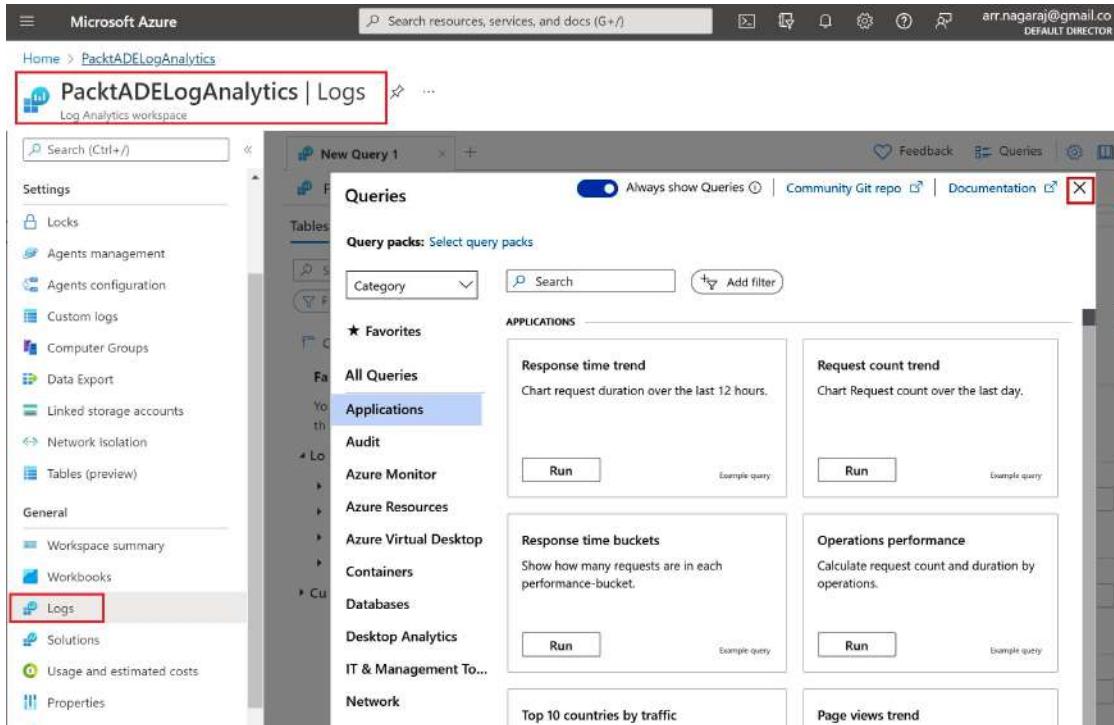


Figure 11.13 – Log Analytics Query

3. Kusto is the language that's used to query Log Analytic Workspace. Copy and paste the Kusto Query provided here. Provide the appropriate values for the Start_Time, End_Time, and DatabaseName variables. Start_Time and End_Time are in UTC:

```

let Start_Time = datetime(2022-05-21 05:00:00);
let End_Time = datetime(2022-05-21 06:00:00);
let DatabaseName = "packtadesqlpool";
SynapseSqlPoolExecRequests
| where TimeGenerated between (Start_Time .. End_Time) and StartTime between (datetime(2000-05-20) .. TimeGenerated)
| where Label != "health_checker"
| where Status contains "Running"
| where _ResourceId endswith DatabaseName
| extend duration_sec = datetime_diff("second", TimeGenerated, StartTime)
| summarize duration_sec = max(duration_sec), Command
    
```

```

= any(Command), Label = any(Label), ResourceClass =
any(ResourceClass), QueryPlan = any(ExplainOutput), Status
= any(Status), Source = any(SourceSystem) by RequestId
| order by duration_sec
| limit 10

```

This script provides the top 10 longest-running active queries for the time specified and provides the duration the query ran for and the resource class it was assigned:

The screenshot shows the Azure Synapse Studio interface with a query editor and a results table.

Query Editor (Top):

```

1 let Start_Time = datetime(2022-05-21 05:00:00);
2 let End_Time = datetime(2022-05-21 06:00:00);
3 let DatabaseName = "packtadesqlpool";
4 SynapseSqlPoolExecRequests
5 | where TimeGenerated >= Start_Time and TimeGenerated <= End_Time and StartTime between (datetime(2000-05-20)..TimeGenerated)
6 | where Label != "health_checker"
7 | where Status contains "Running"
8 | where _ResourceId endswith DatabaseName
9 | extend duration_sec = datetime_diff("second", TimeGenerated, StartTime)
10 | summarize duration_sec = max(duration_sec), Command = any(Command), Label = any(Label), ResourceClass = any(ResourceClass),
    QueryPlan = any(ExplainOutput), Status = any(Status), Source = any(SourceSystem) by RequestId

```

Results Table (Bottom):

duration_sec	RequestId	Command	ResourceClass	Status
> 6	QID2388	Select t1.pid,t1.c1,t2.c2,sum(t2.order_count) FROM dbo.transaction_tbl t1 inner join dbo.transaction_tbl t2 ...	smallrc	Running
> 2	QID2060	SELECT convert(int,rand()) * -1 AS '#'		Running
> 2	QID1989	EXEC sp_set_session_context @key='%', @value='%'		Running
> 2	QID2061	EXEC sp_executesql @sql		Running
> 2	QID2056	SELECT convert(int,rand()) * -1 AS '#'		Running
> 2	QID2059	SELECT tid,transaction_date,order_count,c1,c2 FROM dbo.transaction_tbl where tid = -1	smallrc	Running
> 2	QID2062	SELECT tid,transaction_date,order_count,c1,c2 FROM dbo.transaction_tbl where tid = -1		Running
> 2	QID2057	EXEC sp_executesql @sql		Running
> 2	QID2058	SELECT tid,transaction_date,order_count,c1,c2 FROM dbo.transaction_tbl where tid = -1		Running
> 2	QID2063	SELECT tid,transaction_date,order_count,c1,c2 FROM dbo.transaction_tbl where tid = -1	smallrc	Running

Figure 11.14 – Long-running SQL queries

Perform the following steps to find the highest CPU-consuming Spark jobs:

- First, we need to run a few spark notebooks so that we have some diagnostic data recorded in our Log Analytics workspace. So, download the `sparkpool_notebook1.ipynb` and `sparkpool_notebook2.ipynb` notebooks from <https://github.com/PacktPublishing/Azure-Data-Engineering-Cookbook-2nd-edition/blob/main/chapter11>.
- In **Synapse Studio**, click the develop button (the notebook-like button) on the left. Click on the + symbol, select **Import**, and select the `sparkpool_notebook1.ipynb` file that was downloaded. Similarly, import the `sparkpool_notebook2.ipynb` file.
- Go to the `sparkpool_notebook1` notebook. Select **packtsparkpool** from the **Attach to** dropdown. Then, click **Run all**:

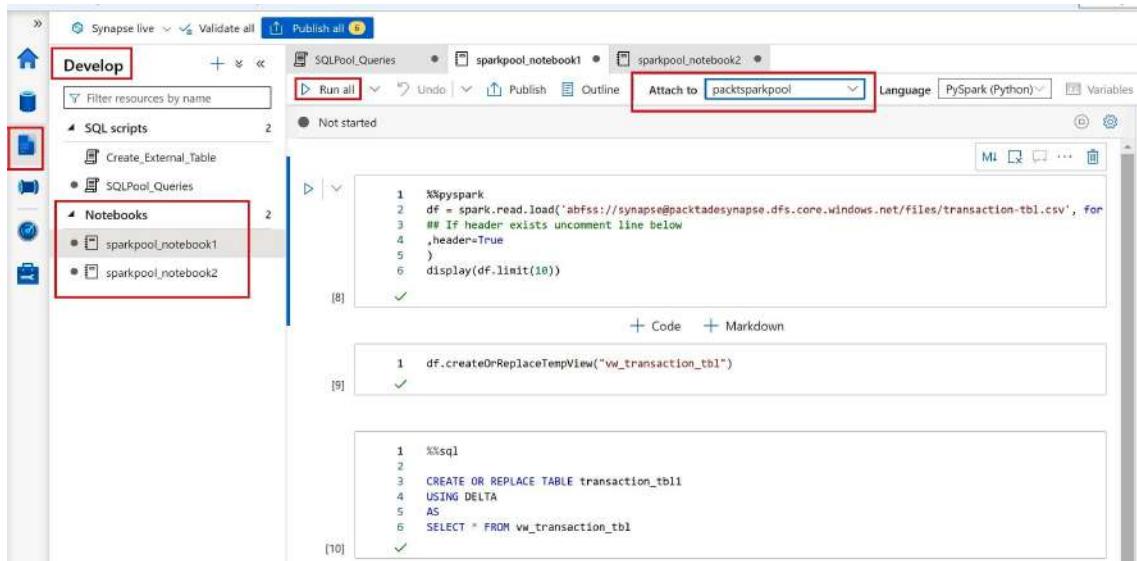


Figure 11.15 – Import notebook

4. Similarly, run the **sparkpool_notebook2** notebook. Both notebooks should finish running in 3 to 5 minutes.
5. Once the run completes, wait 10 to 15 minutes. You will need to wait this long as it can take up to 15 minutes for the notebook's execution data to reflect in the Log Analytics workspace. Go to **PacktADELogAnalytics**, the Log Analytics workspace. Click on **Logs**. Close the **Queries** popup (if it appears) using the close button at the top right. Then, copy and paste the following Kusto script in the new query window:

```

let CpuData =
    SparkMetrics_CL
    | where workspaceName_s == "packtadesynapse" and
clusterName_s == "packtsparkpool"
    | where name_s contains_cs "executor.cpuTime"
    | extend cputime = count_d / 1000000
    | summarize sum(cputime) by
TimeGenerated, applicationName_s;

CpuData
    | summarize cpu_time_ms = max(sum_cputime) by
bin(TimeGenerated, 10m), applicationName_s
    | sort by cpu_time_ms desc

```

```

| project applicationName_s,cpu_time_
ms,bin(TimeGenerated,10m)
| limit 10

```

Set **Time range** to **Last hour** and click the **Run** button. The execution results are shown in the following screenshot:

The screenshot shows the Azure Log Analytics workspace for 'PacktADELogAnalytics'. On the left, the navigation menu is visible with 'Logs' selected. In the center, a 'New Query 1*' tab is open. The query editor contains the following KQL script:

```

3 | where workspaceName_s == "packtadesynapse" and clusterName_s == "packtsparkpool"
4 | where name_s contains_cs "executor.cpuTime"
5 | extend cputime = count_d / 1000000
6 | summarize sum(cputime) by TimeGenerated,applicationName_s;
7
8 CpuData
9 | summarize cpu_time_ms = max(sum(cputime)) by bin(TimeGenerated,10m),applicationName_s
10 | sort by cpu_time_ms desc
11 | project applicationName_s,cpu_time_ms,bin(TimeGenerated,10m)
12 | limit 10

```

The 'Run' button is highlighted with a red box. Below it, the 'Time range' is set to 'Last hour'. The results pane shows a table with the following data:

TimeGenerated [UTC]	applicationName_s	cpu_time_ms
> 5/21/2022, 9:40:00.000 AM	sparkpool_notebook2_packtsparkpool_1653125352	24.657.291
> 5/21/2022, 9:50:00.000 AM	sparkpool_notebook2_packtsparkpool_1653125352	24.657.291
> 5/21/2022, 10:00:00.000 AM	sparkpool_notebook2_packtsparkpool_1653125352	24.657.291
> 5/21/2022, 9:50:00.000 AM	sparkpool_notebook1_packtsparkpool_1653125342	21.480.457
> 5/21/2022, 9:40:00.000 AM	sparkpool_notebook1_packtsparkpool_1653125342	21.480.457
> 5/21/2022, 10:00:00.000 AM	sparkpool_notebook1_packtsparkpool_1653125342	21.480.457

Figure 11.16 – KQL script for the Spark pool's CPU

The script has a `where` clause that has filters against the following three columns:

- `workspaceName_s`: Takes the Synapse workspace's name
- `clusterName_s`: Takes the Spark pool's name
- `name_s`: Filters for CPU time

The `name_s` column contains several key performance metrics related to CPU, memory, data movement, and many more. In this script, we focused on CPU execution time. The script gives the most expensive notebooks for every 10-minute window, measured by CPU time consumed. The `applicationName_s` column reveals the notebook name or the job name that ran on the Spark pool. This script shows that `sparkpool_notebook2` took 3 seconds longer than `sparkpool_notebook1`.

6. I have uploaded another KQL script, `shuffle_operation.txt`, in <https://github.com/PacktPublishing/Azure-Data-Engineering-Cookbook-2nd-edition/blob/main/chapter11> to track Spark jobs or notebooks that moved the most amount of data across the nodes. If you are interested, you can try it out to monitor Spark jobs to maximize data movement via shuffle operations.

How it works...

Configuring a Log Analytics workspace with a Synapse Spark pool and a SQL pool ensures the diagnostic logs are recorded in the Log Analytics workspace. The preceding two scripts that we covered showcase how you could use Kusto scripts in a Log Analytics workspace to find the top SQL queries or Spark jobs. There are additional scripts provided in https://github.com/microsoft/Azure_Synapse_Toolbox/tree/master/Log_Analytics_queries that you can use to explore Log Analytics workspace usage further for Synapse-dedicated SQL pool monitoring.

Creating workbooks in a Log Analytics workspace to visualize monitoring data

Workbooks help you visually explore data stored in Log Analytics workspaces. Workbooks make exploring data stored in a Log Analytics workspace easier since you don't need to write Kusto queries to read the data. In this recipe, we will create two workbooks – one each for monitoring Synapse Spark and SQL pools.

Getting ready

To get started, log in to <https://portal.azure.com> using your Azure credentials:

- Complete the *Configuring a Log Analytics workspace for Synapse dedicated SQL pools* and *Configuring a Log Analytics workspace for Synapse Spark pools* recipes covered earlier in this chapter.
- Download the `SQLPool_Queries.sql` script from https://github.com/PacktPublishing/Azure-Data-Engineering-Cookbook-2nd-edition/blob/main/chapter11/SQLPool_Queries.sql and run it in **Packtadesqlpool**, as explained in *step 1* of the *How to do it...* section of the *Using Kusto queries to monitor Spark and SQL pools* recipe.
- Download the `sparkpool_notebook1.ipynb` and `sparkpool_notebook2.ipynb` notebooks from <https://github.com/PacktPublishing/Azure-Data-Engineering-Cookbook-2nd-edition/blob/main/chapter11>, import them into the **Packtadesynapse** workspace, and execute that workspace against **packtadesparkpool**, as explained in *step 4* of the *How to do it...* section of the *Using Kusto queries to monitor Spark and SQL pools* recipe.

How to do it...

First, let's configure a workbook for monitoring Synapse-dedicated SQL pools:

1. Download the DedicatedSQLPool.workbook file from <https://github.com/PacktPublishing/Azure-Data-Engineering-Cookbook-2nd-edition/blob/main/chapter11>.
2. Log in to portal.azure.com, go to **All resources**, and search for **PacktADELogAnalytics**, the Log Analytics workspace. Under the **General** section, click on **Workbooks**. Click on **Empty** under the **Quick start** section to create an empty workbook:

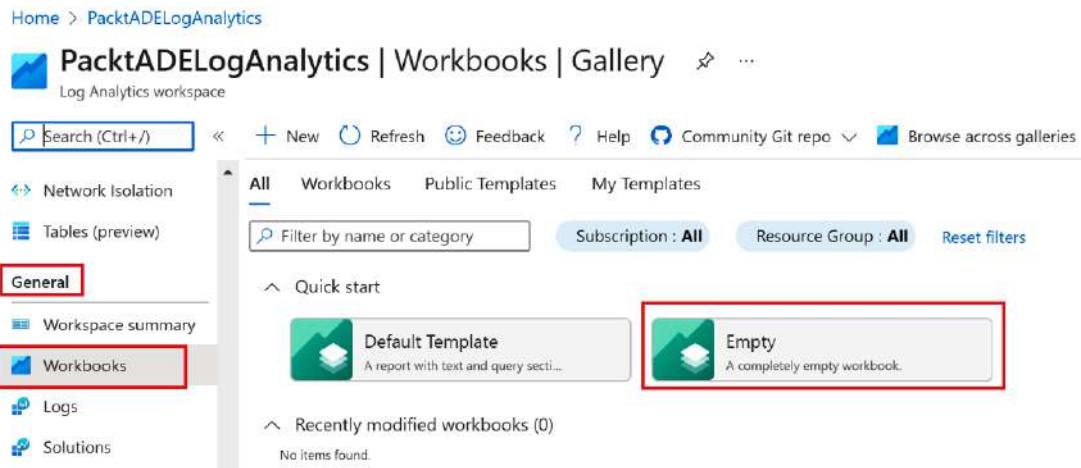


Figure 11.17 – Creating an empty workbook

3. Click on the </> (Advanced Editor) icon on the right:

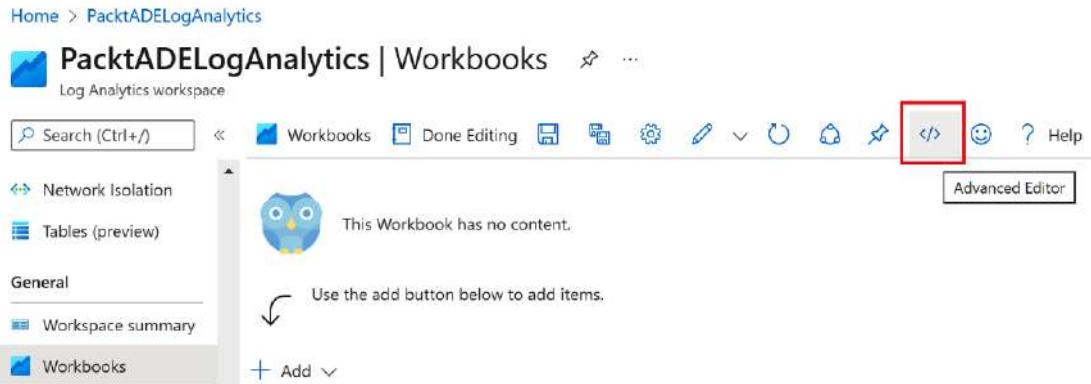


Figure 11.18 – Workbook Advanced Editor

4. Erase whatever content is in the text area and keep it empty:

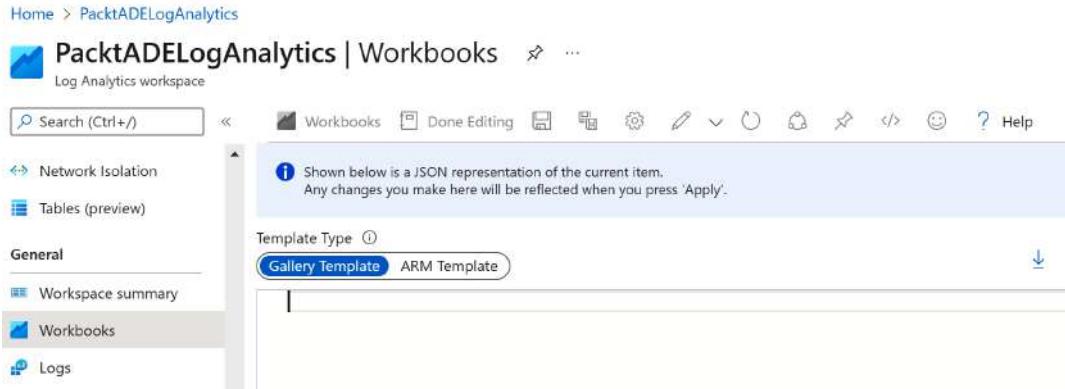


Figure 11.19 – Emptying Advanced Editor

5. Open the DedicatedSQLPool.workbook file you download in step 1 using Notepad. Copy the complete content and paste it into Advanced Editor in the Log Analytics workbook. Click on Apply:

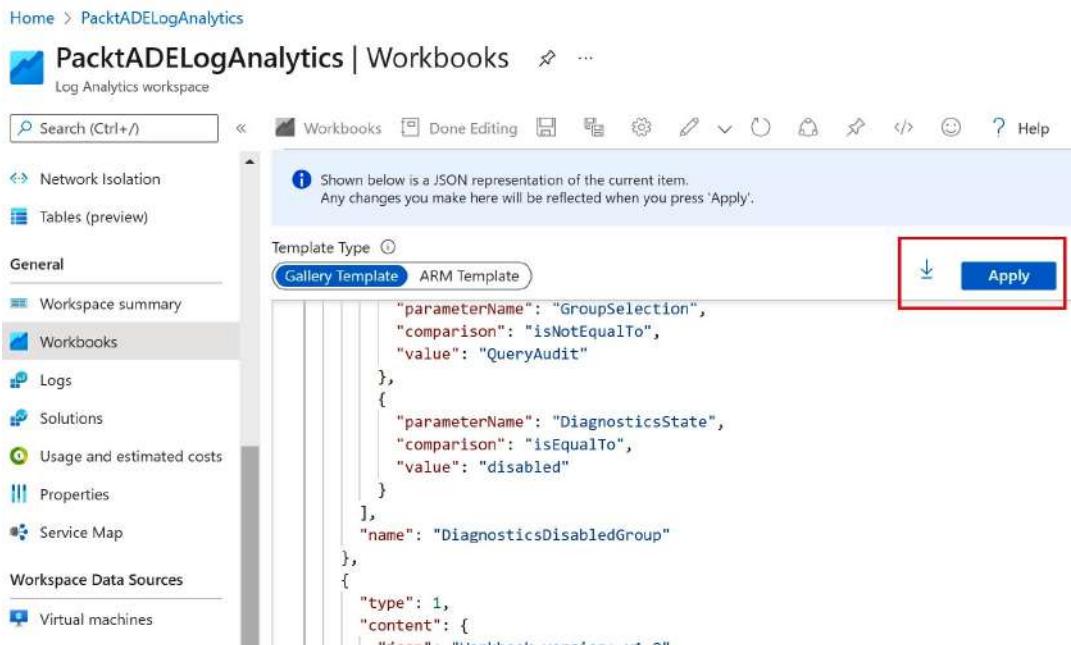


Figure 11.20 – Loading the DedicatedSQLPool workbook

6. Set LogAnalyticsSubscription to be your subscription name, LogAnalyticsWorkspace as PacktADELogAnalytics, and DatabaseResourceName as packtadesynapse/packtadesqlpool. Click the **Done Editing** button:

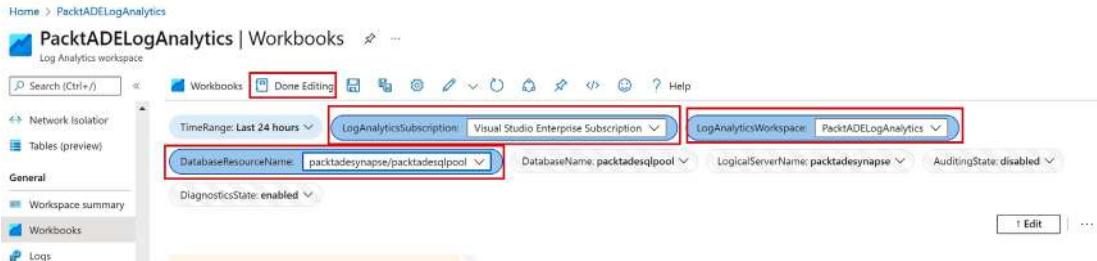


Figure 11.21 – Workbook – Done Editing

7. You should see the Log Analytics data being visualized. Click the **Save** icon at the top. Set **Title** to **DedicatedSQLPool** and click the **Save** button once more:

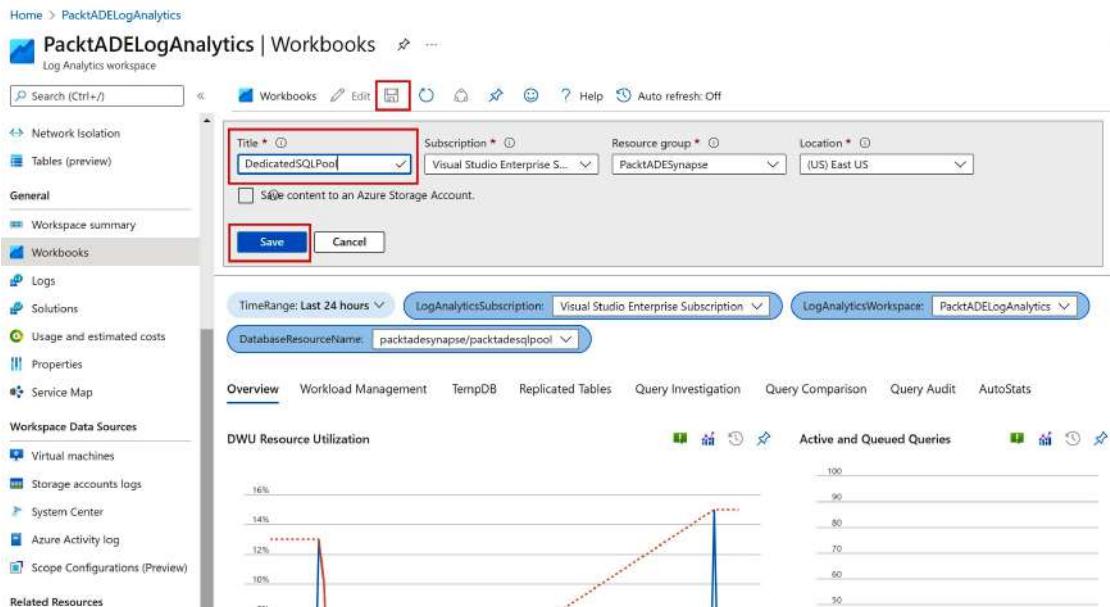


Figure 11.22 – Workbook – Save

8. Let's do a basic performance check using the **DedicatedSQLPool** workbook:
- Observe the DWU usage spike at 9:07 A.M. via the **DWU Resource Utilization** tile in the **Overview** tab:

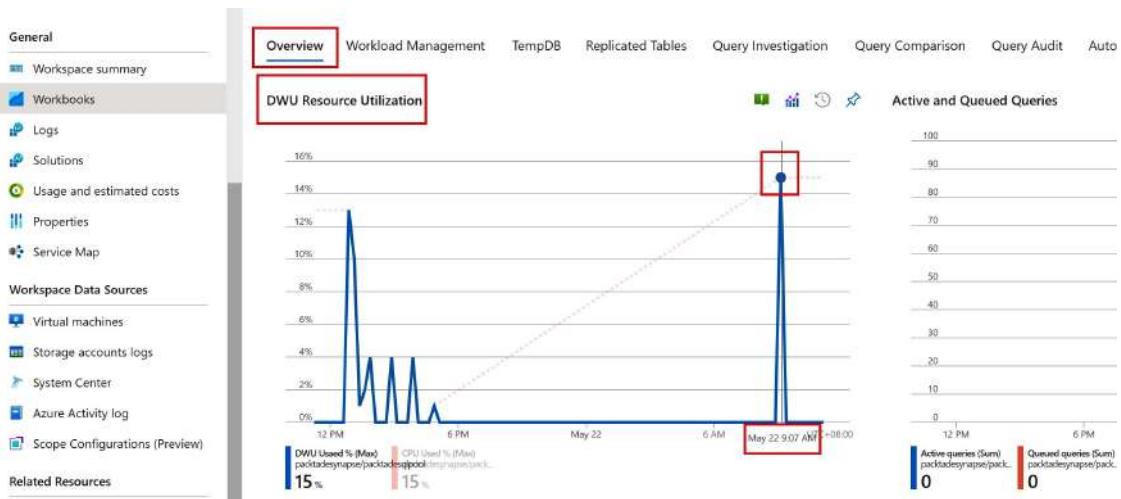


Figure 11.23 – DWU spike

- II. To find the queries that were active at 9:07 A.M., scroll down to the All Queries section in the same tab. Notice that **QID3410** was active. Let's get the query plan for query **QID3410**:

All Queries	Success Only	Failures Only				
Query Completions - use above buttons to filter						
Request_ID ↑ Elapsed Time_min ↑ Start_Time ↑ End_Time ↑ Command ↑ Status ↑ Statement_Ty						
QID3410	0	5/22/2022, 9:07:44 AM	5/22/2022, 9:07:47 AM	Select t1.pid,t1.c1,t2.c2,sum(t2.order_count) FROM dbo.tr...	Completed	Select
QID3409	0	5/22/2022, 9:07:44 AM	5/22/2022, 9:07:44 AM	SELECT tid,transaction_date,order_count,c1,c2 FROM dbo....	Completed	Select
QID3406	0	5/22/2022, 9:07:44 AM	5/22/2022, 9:07:44 AM	SELECT convert(int,rand() * -1) AS '#'	Completed	Select
QID3405	0	5/22/2022, 9:07:44 AM	5/22/2022, 9:07:44 AM	SELECT tid,transaction_date,order_count,c1,c2 FROM dbo....	Completed	Select

Figure 11.24 – All Queries

- III. Switch over to the **TempDB** tab:

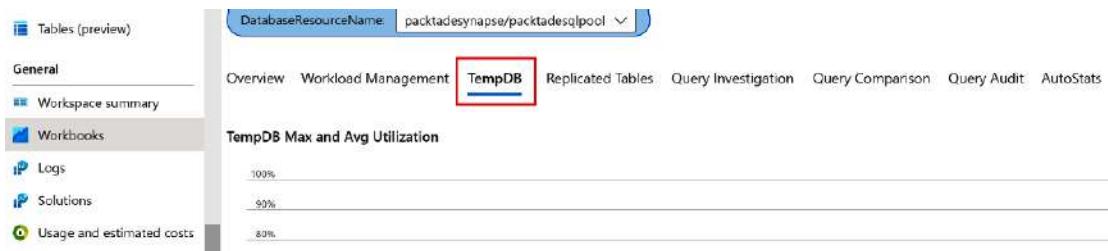


Figure 11.25 – The TempDB tab

- IV. Scroll down to **20 Largest Query Steps by Most Rows Moved**. Let's click on **QID3410**, which we noted down earlier:

20 Largest Query Steps by Most Rows Moved						
Request_ID	Request_Elapsed_Min	ReqStart	ReqEnd	ReqStatus	StepIndex	OperationType
QID3410	0.05	5/22/2022, 9:07:44 AM	5/22/2022, 9:07:47 AM	Completed	5	ShuffleMoveOperation
QID2388	0.08	5/21/2022, 1:12:43 PM	5/21/2022, 1:12:47 PM	Completed	5	ShuffleMoveOperation
QID3009	0.33	5/22/2022, 9:06:35 AM	5/22/2022, 9:06:55 AM	Completed	5	ShuffleMoveOperation
QID2196	0	5/21/2022, 1:12:34 PM	5/21/2022, 1:12:34 PM	Completed	0	ReturnOperation

Figure 11.26 – 20 Largest Query Steps by Most Rows Moved

- V. Scroll down further to the **Query Plan** section. The query plan for **QID3410** will be listed clearly. Here, we can see that **ShuffleMoveOperation** and **ReturnOperation** took the longest:

Selected Query						
Request_ID	ElapsedTime_min	Start_Time	End_Time	Command	Status	
QID3410	0	5/22/2022, 9:07:44 AM	5/22/2022, 9:07:47 AM	Select t1.pid,t1.c1,t2.c2,sum(t2.order_count) FROM dbo.transaction_tbl t1 i...	Completed	

Query Plan									
StepIndex	max_StartTime	max_EndTime	max_RequestId	max_OperationType	max_RowCount	max_Command	max_Status		
0	5/22/2022, 9:07:44 AM	5/22/2022, 9:07:44 AM	QID3410	RandomIDOperation	-1		Complete		
1	5/22/2022, 9:07:44 AM	5/22/2022, 9:07:44 AM	QID3410	OnOperation	-1		Complete		
2	5/22/2022, 9:07:44 AM	5/22/2022, 9:07:44 AM	QID3410	BroadcastMoveOperation	79		Complete		
3	5/22/2022, 9:07:44 AM	5/22/2022, 9:07:44 AM	QID3410	RandomIDOperation	-1		Complete		
4	5/22/2022, 9:07:44 AM	5/22/2022, 9:07:44 AM	QID3410	OnOperation	-1		Complete		
5	5/22/2022, 9:07:44 AM	5/22/2022, 9:07:45 AM	QID3410	ShuffleMoveOperation	79933		Complete		
6	5/22/2022, 9:07:45 AM	5/22/2022, 9:07:47 AM	QID3410	ReturnOperation	79933		Complete		
7	5/22/2022, 9:07:47 AM	5/22/2022, 9:07:47 AM	QID3410	OnOperation	-1		Complete		
8	5/22/2022, 9:07:47 AM	5/22/2022, 9:07:47 AM	QID3410	OnOperation	-1		Complete		

Figure 11.27 – Query Plan

Now, let's create a workbook for monitoring Spark pools.

- Download the `SparkPool.workbook` file from <https://github.com/PacktPublishing/Azure-Data-Engineering-Cookbook-2nd-edition/blob/main/chapter11>.
- Follow steps 2 to 5 to copy the contents of the `SparkPool.Workbook` file to the Advanced Editor area of the new workbook to create Spark pools. Set the workspace's name as `PacktADESynapse` and the Spark pool as `packtsparkpool`. Set `App Livy Id | Name` for any notebook or job run you would like to check. Then, click the **Done Editing** button:



Figure 11.28 – Configuring a Spark workbook

11. Click the **Save** icon. Then, set **Title** to **Sparkpool**. Click the **Save** button once more to save the workbook:

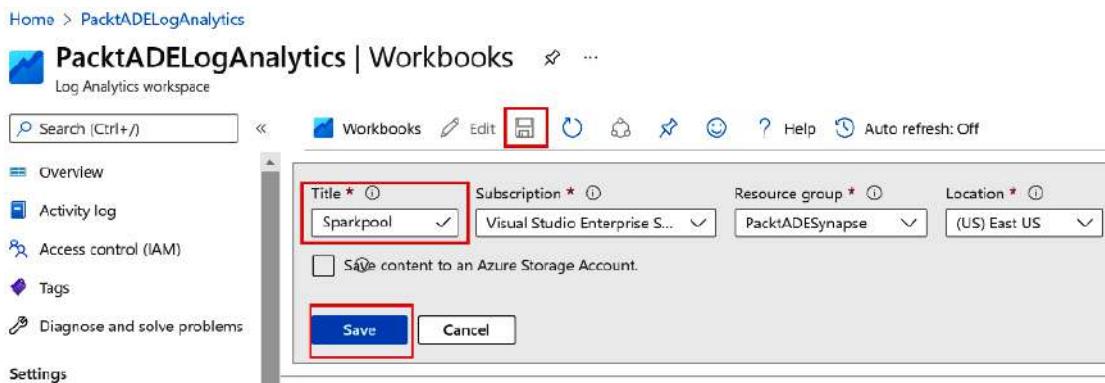


Figure 11.29 – Configuring the Spark workbook

12. Switch to the **Application Summary** tab. The **Application Summary** tab indicates how many executors (or how many worker nodes) were used to run the notebook and how many internal jobs were executed and their statuses:

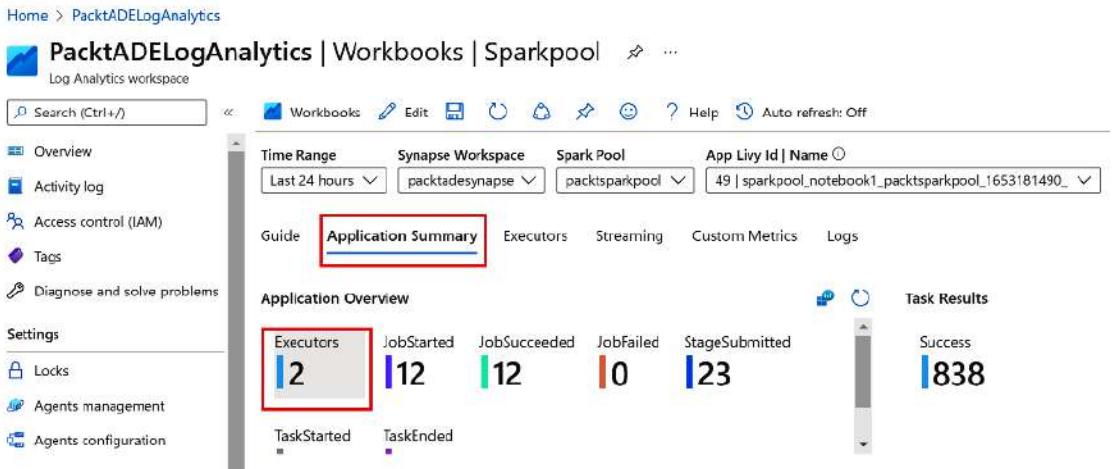


Figure 11.30 – Sparkpool workbook overview

13. Click on the **Executors** tab. The **Executors** tab provides insights into CPU usage, memory usage, and data processed by each node:

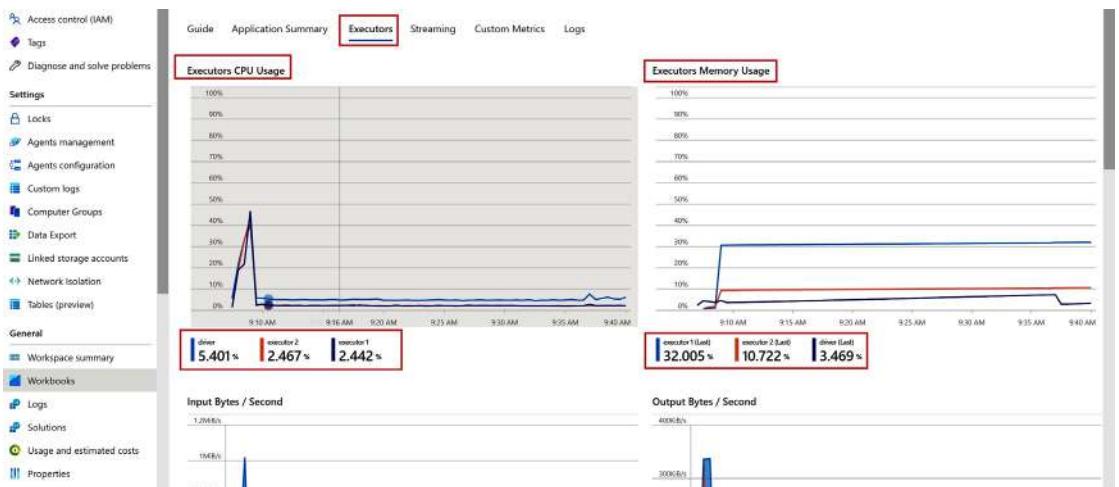


Figure 11.31 – Sparkpool workbook overview

How it works...

Workbooks are like templated Kusto scripts combined with visuals that cover all the key metrics required for monitoring the health of Azure services. Workbooks use Kusto scripts to read the data in a Log Analytics workspace and present it in simple visuals for easier interpretation. The workbooks

used in this recipe can be found at https://github.com/microsoft/Azure_Synapse_Toolbox/tree/master/Monitor_Workbooks, which is part of the official GitHub page maintained by Microsoft. This GitHub page contains other workbooks for serverless SQL pools and Synapse pipelines. We recommended that you explore these while following the configuration method explained in this recipe.

Monitoring table distribution, data skew, and index health using Synapse DMVs

In distributed databases such as Synapse dedicated SQL pools, the table is distributed across multiple nodes by design. When the rows in the table are not evenly distributed across the nodes, data distribution is said to be skewed. Data skew scenarios can have an impact on query performance. This recipe will provide a script based on Synapse **Dynamic Management Views (DMVs)** that you can use to monitor table skew.

Tables in a dedicated SQL pool have a column store index created by default. Column store indexes store the rows of the table in columnar format, which is optimized for processing analytics workloads. Each column store index in a table is subdivided into segments. A column store segment can be of three states – **Open**, **Closed**, or **Compressed**. For the column store index to be effective, its segments need to meet the following conditions:

- The number of segments in an open or closed state should be minimal.
- Each column store segment should have at least 100,000 rows.
- Segments in a compressed state should have at least a million rows.

In this recipe, we will develop a script to check for these conditions and verify the column store index's health.

Getting ready

To get started, log in to <https://portal.azure.com> using your Azure credentials:

- Create a Synapse Analytics workspace, as explained in the *Provisioning an Azure Synapse Analytics workspace* recipe of *Chapter 8, Processing Data Using Azure Synapse Analytics*.
- Download the `transaction-tbl.csv` file from <https://github.com/PacktPublishing/Azure-Data-Engineering-Cookbook-2nd-edition/tree/main/chapter10>. In the Synapse Analytics workspace, create a folder named `files` in the Data Lake account attached to it. Upload the `transaction-tbl.csv` file to the `files` folder. For detailed screenshots for a similar task, follow *steps 1 to 4* in the *How to do it...* section of the *Analyzing data using a serverless SQL pool* recipe of *Chapter 8, Processing Data Using Azure Synapse Analytics*.

- Complete the *Loading data into a dedicated SQL pool using PolyBase and T-SQL* recipe of *Chapter 10, Building the Serving Layer in Azure Synapse SQL Pool*, to create a dedicated SQL pool named `packtadesqlpool`, an external table named `dbo.ext_transaction_tbl`, and a dedicated SQL pool table named `dbo.transaction_tbl`. Alternatively, you may use the `Create_External_Table.sql` script at <https://github.com/PacktPublishing/Azure-Data-Engineering-Cookbook-2nd-edition/tree/main/chapter10> to create the `dbo.ext_transaction_tbl` and `dbo.transaction_tbl` tables.

How to do it...

In this recipe, we will be performing two major tasks:

- Detecting tables with skewed data distributions using Synapse DMVs
- Detecting tables with poor column store indexes

Perform the following steps to detect tables with skewed data:

1. First, let's create a skewed table. Log in to `portal.azure.com`, go to **All resources**, and search for `packtadesynapse`. Click on the workspace. Then, click on **Open Synapse Studio**. Click the develop button (the notebook-like button) on the left. Click the **+** button and select **SQL Script**. Finally, connect to the `packtadesqlpool` database and run the following script:

```
CREATE table dbo.transaction_tbl_skew WITH (DISTRIBUTION
= HASH(pid_skew))
AS
SELECT tid,sid, case when pid > 2 then 8 else pid end as
pid_skew,
transaction_date,
order_count,c1,c2
FROM dbo.transaction_tbl
```

This script creates a table that's distributed based on the `pid_skew` column, which will mostly contain a value of 8, causing the table to be distributed in an uneven or skewed fashion. The script's execution is shown in the following screenshot:

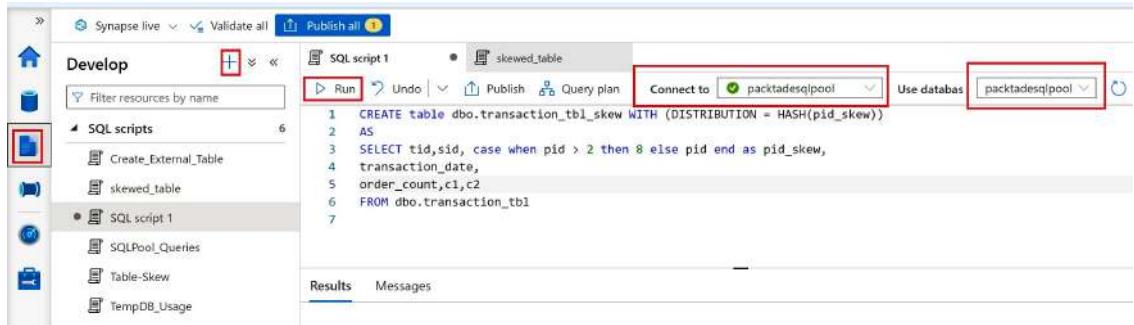


Figure 11.32 – Creating a skewed table

Download the `Table-Skew.sql` file from <https://github.com/PacktPublishing/Azure-Data-Engineering-Cookbook-2nd-edition/blob/main/chapter11/Table-Skew.sql>. Open Synapse Studio and click the develop button (the notebook-like button) on the left. Click on the + symbol and select **Import**:

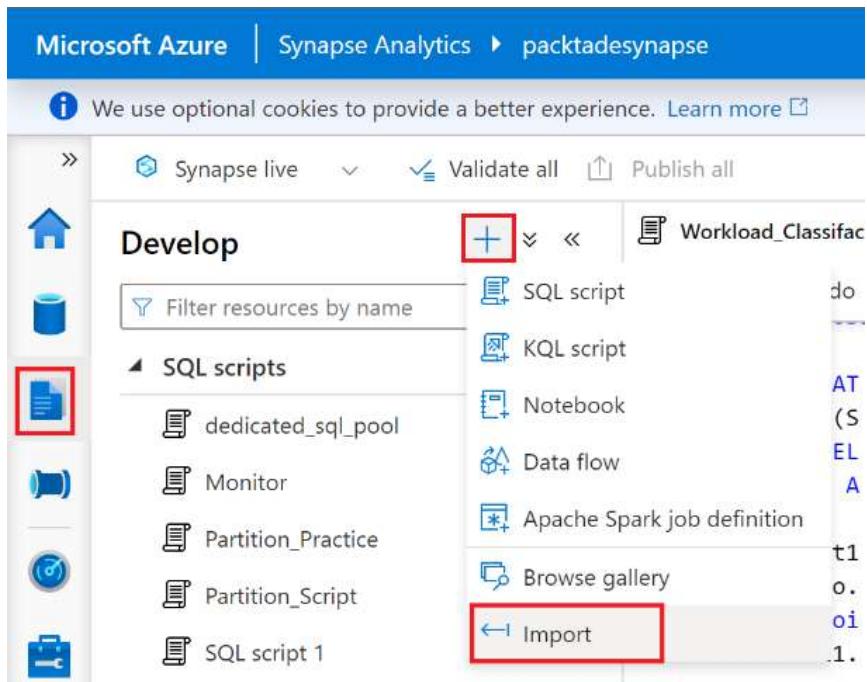


Figure 11.33 – Importing the SQL script

2. Select the `Table_skew.sql` file you downloaded in step 2. Connect to the `packtadesqlpool` database and run the script, as shown in the following screenshot:

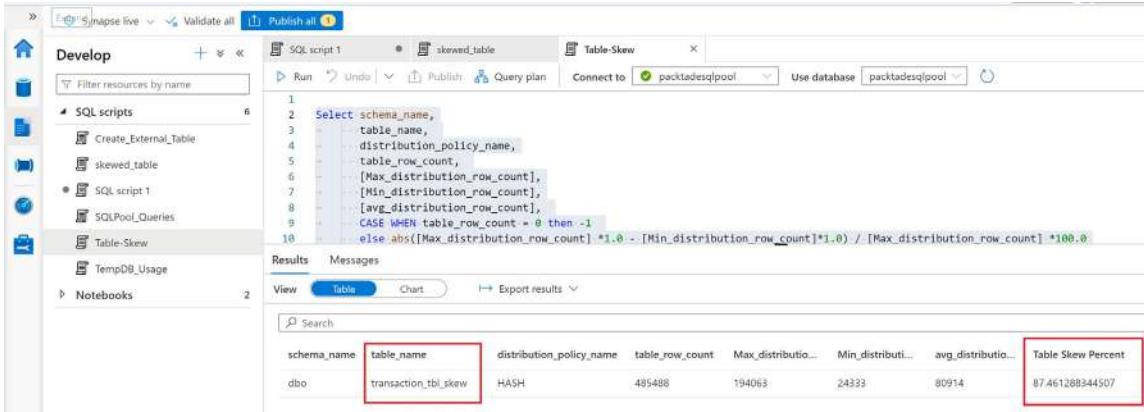


Figure 11.34 – Table skew

A table in a Synapse-dedicated SQL pool will be split into a maximum of 60 distributions (parts) by default, irrespective of the number of nodes in the Synapse data warehouse instance (for example, if the Synapse instance has four nodes, each node will have 15 distributions of the table; if the Synapse instance has 10 nodes, each node will have six distributions). The script uses the `sys.dm_pdw_nodes_db_partition_stats` DMV, which contains several rows in each distribution. To measure the skew, the script compares the lowest and highest row count per distribution using the following formula:

$$\text{Table Skew Percentage} = \frac{(\text{Highest row count per distribution} - \text{Lowest row count per distribution}) * 100}{\text{Highest row count at a distribution}}$$

The higher *Table Skew Percentage* is, the bigger the difference in row count across distributions, and hence the bigger the table skew will be. For tables with *Table Skew Percentage* over 25%, verify the choice of distribution column and recreate the table using a different distribution column, if required.

Now, let's explore a script for checking the quality of the column store index:

1. Download the `Clustered_Index_Health_Check.sql` file from https://github.com/PacktPublishing/Azure-Data-Engineering-Cookbook-2nd-edition/blob/main/chapter11/Clustered_Index_Health_Check.sql. Open Synapse Studio and click the develop button (the notebook-like button) on the left. Then, click on the + symbol and select **Import**:

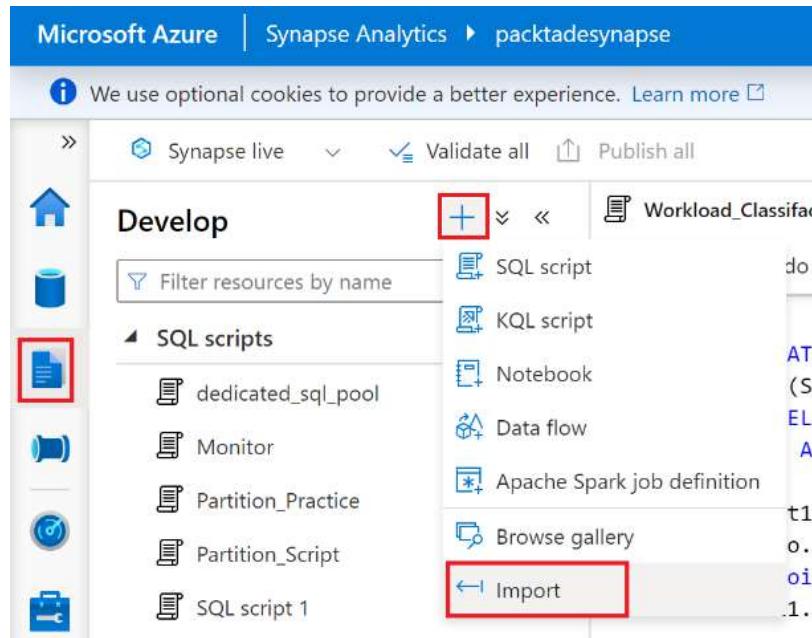


Figure 11.35 – Importing the SQL script

2. Select the `Clustered_Index_Health_Check.sql` file you downloaded in *step 1*. Connect to the `packtadesqlpool` database and run the script, as shown in the following screenshot:

```

1 select schema_name(t.schema_id) [schema_name], t.name as Table_Name,
2 Avg(total_rows) as Average_Rows_Per_Segment,
3 Count(*) as Total_segments,
4 Case when state_desc = 'COMPRESSED' and total_rows < 100000 then 1 else 0 end ) as Not_Optimized_Segments,
5 Sum(Case when state_description = 'OPEN' then 1 else 0 end) as Open_Segments,
6 Sum(Case when state_description = 'CLOSED' then 1 else 0 end) as Closed_Segments ,
7 Case when Avg(total_rows) < 100000 then 'Table doesn''t have enough rows for columnstore index. Consider moving to heap (table without index), if the table
8 When Sum(Case when state_description = 'CLOSED' then 1 else 0 end) > 10 then 'Many segments in Closed State. Run Alter table (table name) Reorganize to move
9 When Sum(Case when state_description = 'COMPRESSED' and total_rows < 100000 then 1 else 0 end ) > 0 then 'Many sub optimal segments found. Recompress the t
10 When Sum(Case when state_description = 'OPEN' then 1 else 0 end) > 10 then 'Too many open segments suggest data loading across partitions. Double check the t
11 Sum(total_rows) as Row_Count
12 FROM sys.pdm_nodes_column_store_row_groups rg

```

schema_name	Table_Name	Average_Rows_Per_Segment	Total_segments	Not_Optimize...	Open_Segments	Closed_Segme...	Recommendation	Row_Count
dbo	transaction_tbl	4045	60	0	60	0	Table doesn't have enou...	242744
dbo	transaction_tbl_skew	80914	3	1	2	0	Table doesn't have enou...	242744

Figure 11.36 – Column store health check script

This script identifies that the tables are too small to be stored using the column store format as they only contain a few thousand rows per segment. The script recommends that the table be converted into a heap table.

The script identifies tables with poor-quality column store indexes and offers recommendations to fix them. The script checks for the following conditions and offers the corresponding recommendations:

Condition Verified	Recommendation	Reason
Checks if the average rows per segment are less than 100,000 rows	Make the table into a heap table, which will store the table without the column store index.	Column store indexes are ideal for large tables with at least a few million rows in total. For small tables, column store indexes would be inefficient.
Checks if there are more than 10 segments that are in an open state	Verify if the table is partitioned, causing too many small segments, and fix the partition.	When the state of the column store segment is open, it remains in an uncompressed format, offering sub-optimal performance for analytic queries. If there are more than 1 million rows per segment, the engine would automatically compress the segment. It is recommended that you verify the reason for small segments – for example, due to the table partitioning strategy or data skewness.
Checks if there are more than 10 segments in a closed state	Run <code>ALTER TABLE <Table name> Reorg</code> to fix it.	If a segment is in a closed state, it implies it is ready to be compressed and is waiting for the system's background process to move it to a compressed state. You could force the background process to compress by performing a table reorg operation.
Checks if any compressed segments have less than 1 million rows (non-optimized segments)	Recreate the table using the <code>Create TABLE AS</code> command or rebuild the index with a higher resources class/more resources. You could scale up the DWU units of synapse the instance during the rebuild operation and scale them down once you've finished.	A compressed segment with less than 1 million rows offers sub-optimal performance. This is possible if there were not enough resources when the table was created or when the index was rebuilt.

Building monitoring dashboards for Synapse with Azure Monitor

By default, Azure Monitor stores the key performance counters of Synapse SQL pools and Sparks pool. Creating a dashboard using Azure Monitor's performance counters allows you to observe all the key performance counters at a glance and deduct any health-related issues on the Synapse workspace.

Getting ready

The prerequisites for this recipe are as follows:

- Create a Synapse Analytics workspace, as explained in the *Provisioning an Azure Synapse Analytics workspace* recipe of *Chapter 8, Processing Data Using Azure Synapse Analytics*.
- Complete the *Loading data into a dedicated SQL pool using PolyBase and T-SQL* recipe of *Chapter 10, Building the Serving Layer in Azure Synapse SQL Pool* to create a dedicated SQL pool named `packtadesqlpool`, an external table named `dbo.ext_transaction_tbl`, and a dedicated SQL pool table named `dbo.transaction_tbl`.
- Download the `SQLPool_Queries.sql` script from https://github.com/PacktPublishing/Azure-Data-Engineering-Cookbook-2nd-edition/blob/main/chapter11/SQLPool_Queries.sql. Run the script, as explained in *step 1* of the *How to do it...* section of the *Using Kusto queries to monitor SQL and Spark pools* recipe.
- Create a Spark pool cluster, as explained in the *Provisioning and configuring Spark pools* recipe of *Chapter 8, Processing Data Using Azure Synapse Analytics*.
- Download the `sparkpool_notebook1.ipynb` and `sparkpool_notebook2.ipynb` notebooks from <https://github.com/PacktPublishing/Azure-Data-Engineering-Cookbook-2nd-edition/blob/main/chapter11>, import them into the **Packtadesynapse** workspace, and execute it against **packtadesparkpool**, as explained in *step 4* of the *How to do it...* section of the *Using Kusto queries to monitor Spark and SQL pools* recipe.

How to do it...

Let's create a dashboard that will monitor the health of Synapse SQL pools and Spark pools by performing the following steps:

1. Log in to `portal.azure.com`, go to **All resources**, and search for `packtadesqlpool`. Search for **Metrics** under **Monitoring** and click on it. In the drop-down for **Metric**, select **DWU used percentage**:

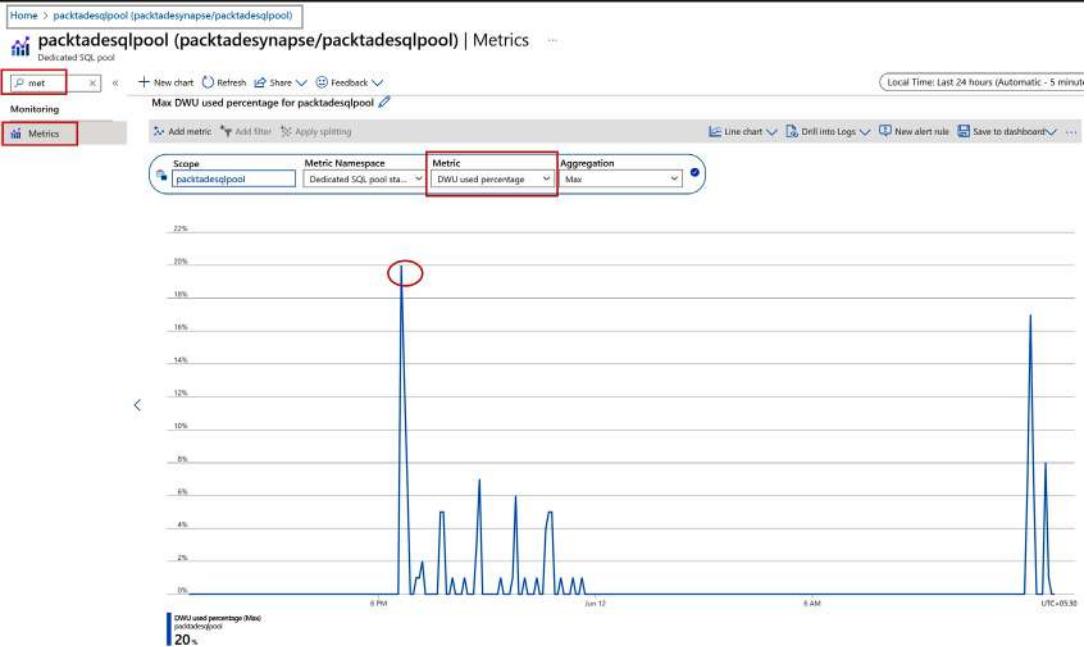


Figure 11.37 – DWU used percentage

The **DWU used percentage** metric indicates the percentage of total **Datawarehouse Units (DWUs)** used at a particular time. **DWU used percentage** is the key metric to identify if the data warehouse can manage the workload. A value of 100 for **DWU used percentage** would indicate that the data warehouse is running at its fullest capacity.

2. At the top right, click on **Save to dashboard** and select **Pin to dashboard**:

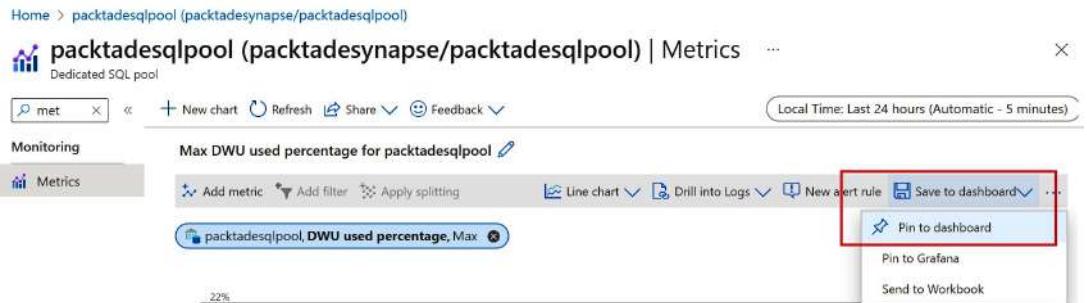


Figure 11.38 – Pin to dashboard

- Click on the **Create new** tab. Select the type as **Shared**. Then, set **Dashboard name** to **SynapseMonitoring**. Uncheck the **Publish to 'dashboards' resource group** option and set **Resource group** to **PacktADESynapse**, the same resource group we used to create the Synapse Analytics workspace. Then, click on **Create and pin**:

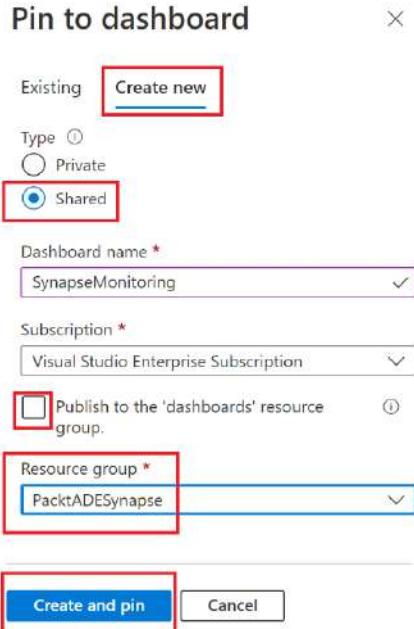


Figure 11.39 – Create and pin

- Click on **+ New chart**. Set **Metric** to **Connections** and click on **Save to dashboard**:

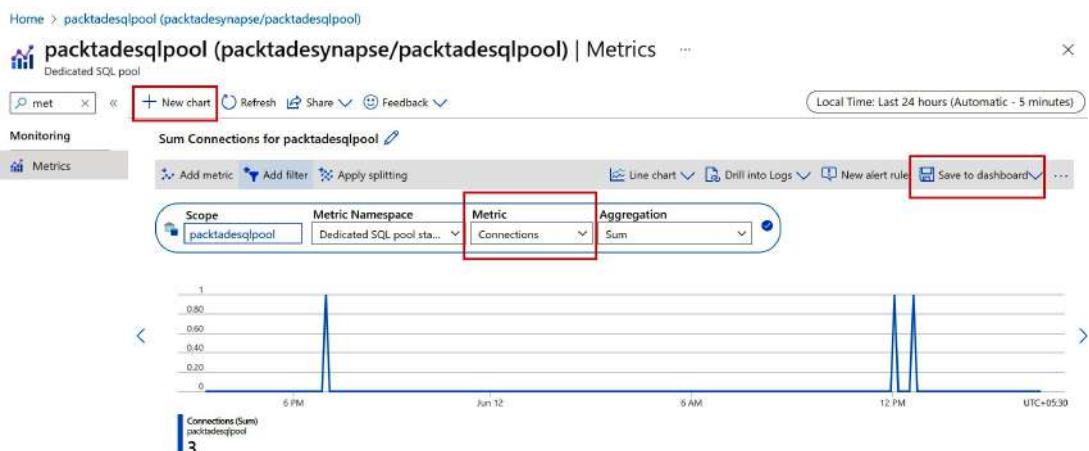


Figure 11.40 – Adding a new chart

5. Select **Pin to dashboard:**

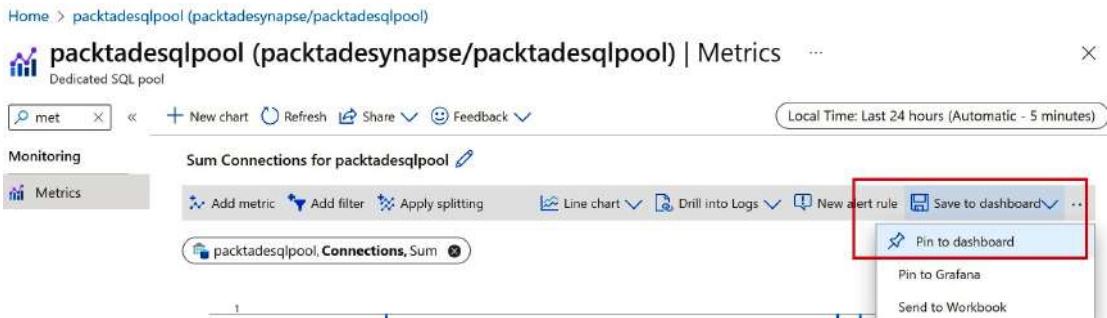


Figure 11.41 – Pin to dashboard

6. Select the **Existing** tab and set **Type** to **Shared**. Set **Dashboard** to **SynapseMonitoring**, the same dashboard we created in *step 3*. Then, click on **Pin**:

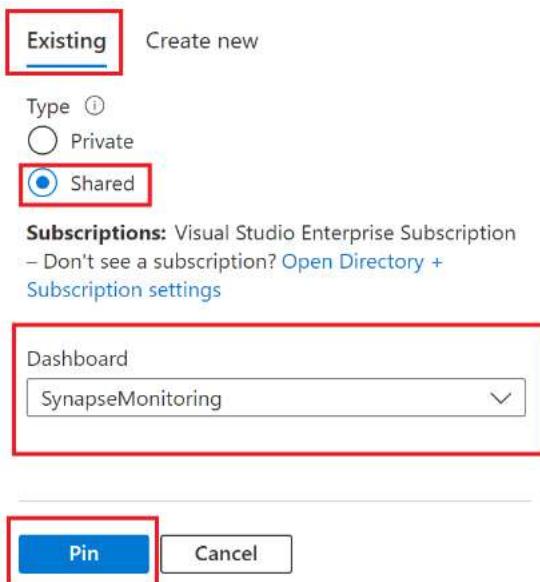


Figure 11.42 – Pin to dashboard

7. Repeat *steps 4* to *6* to create new charts for the following metrics and add them to the **SynapseMonitoring** dashboard:

- **Queued queries:** To track queries that are queued due to a lack of concurrency slots/resources
- **Active queries:** To track the number of active queries in the dedicated SQL pool

- **Memory used percentage:** To track the total memory used in the Synapse-dedicated SQL pool
 - **Local tempdb used percentage:** To track the tempdb usage percentage in the dedicated SQL pool
8. Now, let's add Sparkpool metrics to the dashboard. Go to **All resources** in the Azure portal and search for **packtsparkpool**. Search for **Metrics** under **Monitoring** and click on it. In the drop-down for **Metric**, select **vCores allocated**. Click on **Save to dashboard** and pin it to the **SynapseMonitoring** dashboard, as explained in *steps 4 to 7*:

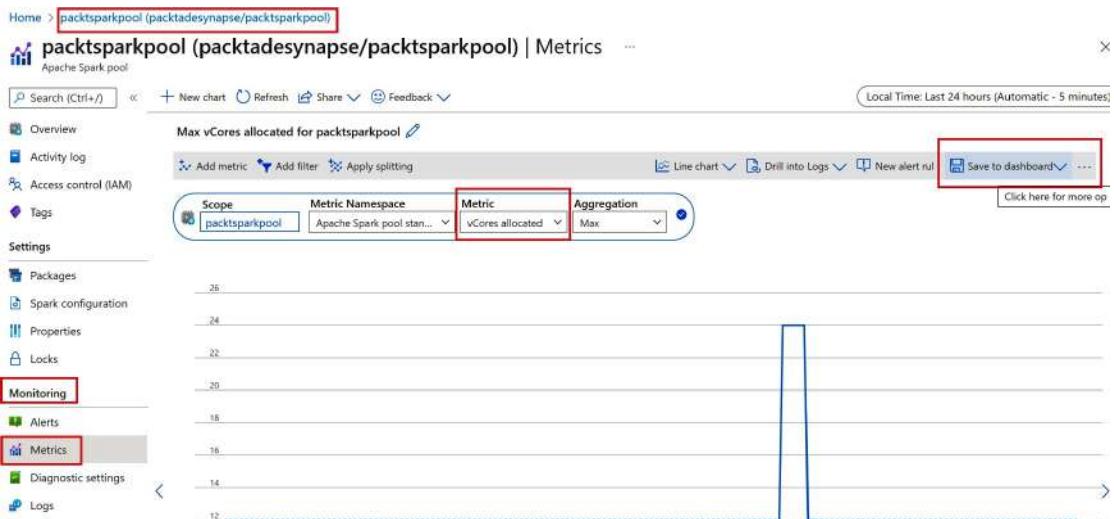


Figure 11.43 – vCores allocated

9. Repeat *steps 4 to 7* to create new charts for the following Sparkpool metrics and add them to the **SynapseMonitoring** dashboard:
- **Active Apache Spark Pool application:** To track the number of active Spark jobs
 - **Memory Allocated:** To track the memory used by the Spark pool
10. Go to **All resources** in the Azure portal and search for **SynapseMonitoring**. A resource of the **Shared Dashboard** type in the **PacktAdeSynapse** resource group should be listed. Click on it:

The screenshot shows the Azure portal's 'All resources' view. At the top, there are several filters: 'Subscription == all', 'Resource group == all', 'Type == all', and 'Location == all'. A red box highlights the 'SynapseMonitoring' filter. Below the filters, there is a summary card for 'Unsecure resources' with a shield icon and the number '0'. To the right, there is a dropdown for 'No grouping' and a link to 'List view'. The main table lists a single resource:

Name	Type	Resource group	Location	Subscription
f0d7f4a0-8145-47cb-a68d-92d10c14227c (SynapseMonitoring)	Shared dashboard	PacktADESynapse	East US	Visual Studio Enterprise Subscription

Figure 11.44 – Dashboard

11. Click on **Go to dashboard**:

This screenshot shows the settings page for a shared dashboard named 'f0d7f4a0-8145-47cb-a68d-92d10c14227c (SynapseMonitoring)'. On the left, there is a sidebar with options: 'Overview' (selected), 'Activity log', 'Access control (IAM)', and 'Tags'. A search bar at the top has a red box around it. On the right, there are two main sections: 'Name' containing the value 'f0d7f4a0-8145-47cb-a68d-92d10c14227c (SynapseMonitoring)' with a copy icon, and 'Resource type' containing 'Microsoft.Portal/dashboards' with a copy icon.

Figure 11.45 – Go to dashboard

12. All the charts we added will appear. Click on **Edit**:

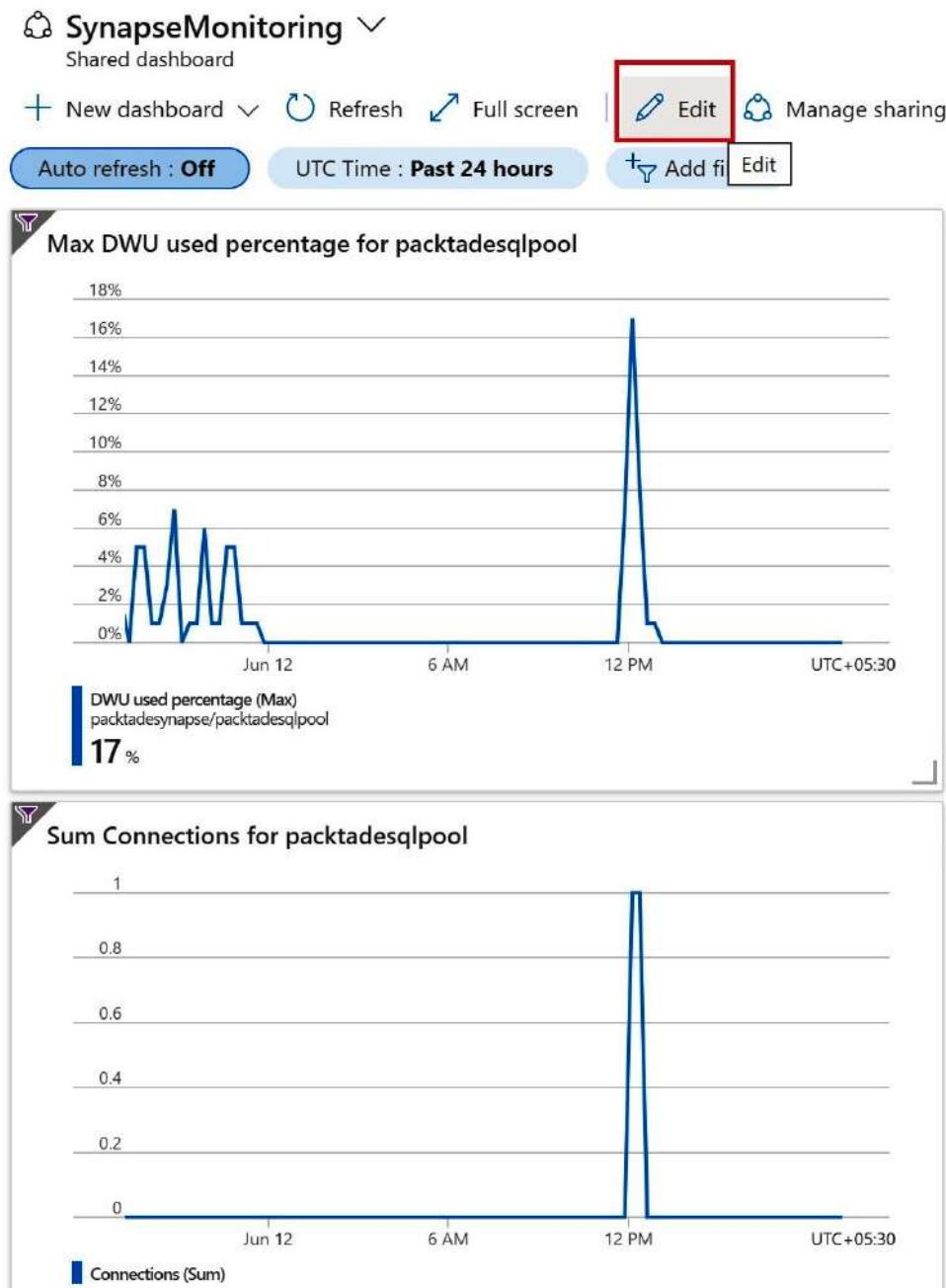


Figure 11.46 – Editing the dashboard

13. Close the Tile Gallery area:

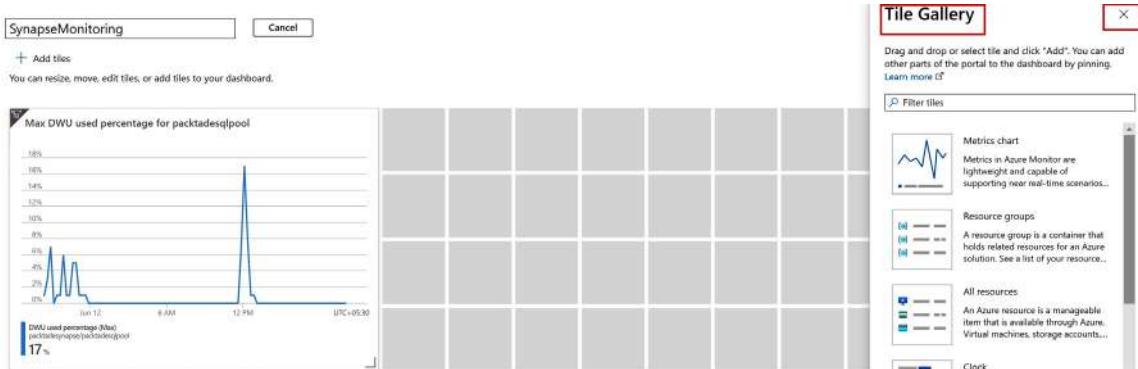


Figure 11.47 – Closing the Tile Gallery area

14. Resize, drag, and reposition the charts based on their importance and the relevance of their metrics. Then, click **Save**:

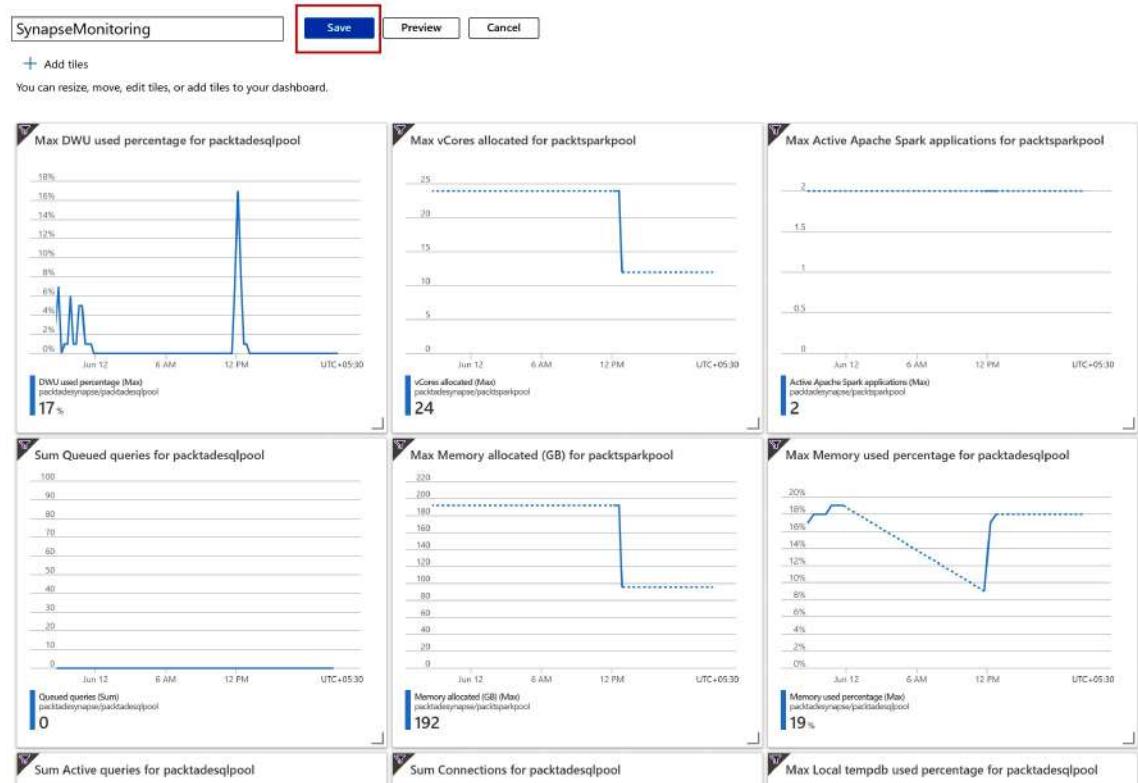


Figure 11.48 – Saving the dashboard

15. Click on **Auto refresh: Off** and set it to **Every 5 minutes**:

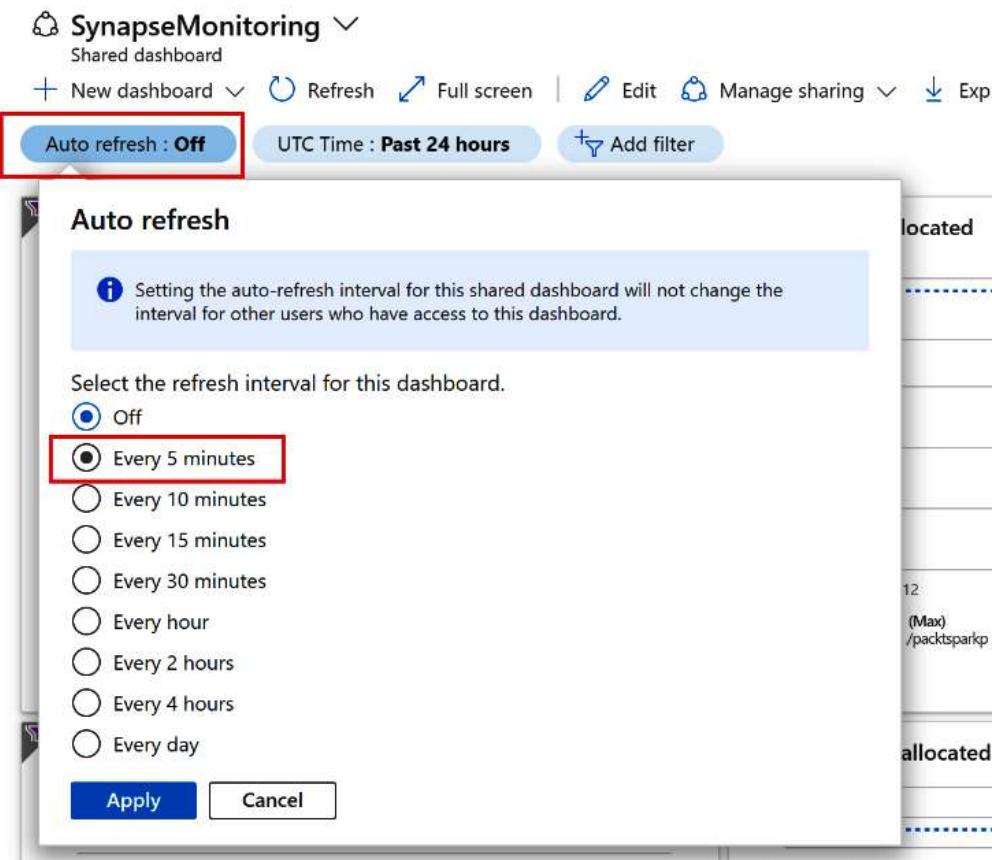


Figure 11.49 – Choosing when to refresh the dashboard

16. The dashboard is ready to be consumed. You may click on the **Manage sharing** button and then select the **Access control** option to grant permission to other users in your organization to view or edit the dashboard:



Figure 11.50 – Access control dashboard

How it works...

By default, Azure Monitor collects the health metrics of all Azure services and stores them for 30 days. Health metrics collected by Azure Monitor can be consolidated and viewed using dashboards. Having the key metrics across services in one place helps you correlate performance problems too. Setting the refresh frequency helps keep the dashboard updated automatically. Dashboards can be easily shared with anyone in your organization.

12

Optimizing and Maintaining Synapse SQL and Spark Pools

In the previous chapter, we learned how to monitor Synapse dedicated SQL and Spark pools. As a natural follow-up to that, in this chapter, we will cover techniques that can optimize the performance of Synapse SQL and Spark pools. In addition to optimization techniques, we will also share maintenance techniques such as pausing Synapse dedicated SQL pools and configuring a longer backup retention duration for Synapse dedicated SQL pools.

By the end of this chapter, you will have learned how to optimize Synapse dedicated SQL pool by reading a query plan and fixing its distribution, building a replication cache, and configuring result set caching. You will also know how to improve Spark pool performance using Z-ordering and partitioning techniques. In addition to optimization techniques, you will have learned about various maintenance techniques to automatically pause Synapse dedicated SQL pool during quiet periods, configure a backup retention that is longer than the default 7 days available in Synapse dedicated SQL pool, and run maintenance operations such as `OPTIMIZE` and `VACUUM`.

In this chapter, we'll cover the following recipes:

- Analyzing a query plan and fixing table distribution
- Monitoring and rebuilding a replication table cache
- Configuring result set caching in Azure Synapse dedicated SQL pool
- Auto pausing Synapse dedicated SQL pool
- Configuring longer backup retention for a Synapse SQL database
- Optimizing Delta tables in a Synapse Spark pool lake database
- Optimizing query performance in Synapse Spark pools

Technical requirements

For this chapter, you will need the following:

- A Microsoft Azure subscription
- PowerShell

Analyzing a query plan and fixing table distribution

Synapse dedicated SQL Pool is a distributed database engine that is comprised of one control node and one or more compute nodes. A control node is like the brain of the database engine and all the queries that are submitted by client applications are received by the control node. The control node splits the work that needs to be done to process the query and assigns it to compute nodes in Synapse dedicated SQL Pool. The compute nodes process the work assigned to them and return the partial results to the control node. The control node combines the results from compute nodes, performs any additional processing, and returns the final output to the client application. The number of compute nodes depends on the **data warehouse units (DWUs)** of the Synapse dedicated SQL pool, with SQL pools with higher DWUs getting more compute nodes. For example, a DWU1500 SQL pool has three compute nodes, while a DWU30000 SQL pool has 60 compute nodes. The architecture of a Synapse dedicated SQL pool with three compute nodes (DWU1500c) can be seen in the following diagram:

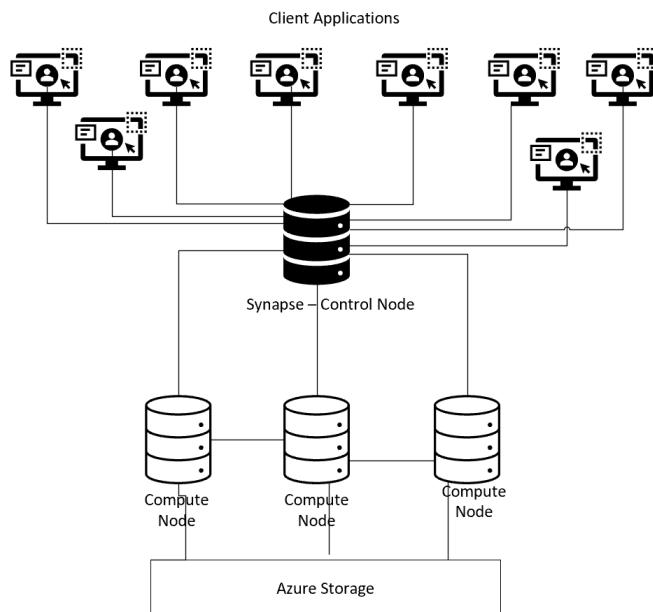


Figure 12.1 – Synapse dedicated SQL pool architecture

As the data is processed in multiple compute nodes, the engine moves the data across nodes to process the queries submitted. Typically, data movement across nodes is the most expensive operation in distributed query processing. Data movement can be reduced by distributing the data across nodes correctly, which can be achieved by selecting the correct distribution strategy for the table.

In this recipe, we will analyze a query plan, identify the data movement, and change the table distribution to improve query performance.

Getting ready

To get started, log into <https://portal.azure.com> using your Azure credentials.

Create a Synapse Analytics workspace, as explained in the *Provisioning an Azure Synapse Analytics workspace* recipe of *Chapter 8, Processing Data Using Azure Synapse Analytics*.

Complete the *Loading data into a dedicated SQL pool using PolyBase and T-SQL* recipe of *Chapter 10, Building the Serving Layer in Azure Synapse SQL Pool*, to create a dedicated SQL pool named `packtadesqlpool`, an external table named `dbo.ext_transaction_tbl`, and a dedicated SQL pool table named `dbo.transaction_tbl`. Alternatively, to create the `dbo.ext_transaction_tbl` and `dbo.transaction_tbl` tables, you may use the `Create_External_Table.SQL` script available at <https://github.com/PacktPublishing/Azure-Data-Engineering-Cookbook-2nd-edition/tree/main/chapter10>.

How to do it...

1. Log into portal.azure.com, go to **All resources**, and search for `packtadesynapse`, the Synapse Analytics workspace you created in the *Provisioning an Azure Synapse Analytics workspace* recipe of *Chapter 8, Processing Data Using Azure Synapse Analytics*. Click on the workspace, then **Open Synapse Studio**:

The screenshot shows the Azure portal interface for a Synapse workspace named 'PacktADESynapse'. The left sidebar contains navigation links for Overview, Activity log, Access control (IAM), Tags, Diagnose and solve problems, Settings (Azure Active Directory, Properties, Locks), Analytics pools (SQL pools, Apache Spark pools, Data Explorer pools (preview)), Security, and Encryption. The main content area displays the 'Essentials' section with details such as Resource group (PacktADESynapse), Status (Succeeded), Location, Subscription (Visual Studio Ultimate with MSDN), Subscription ID, Managed virtual network (Yes), Managed Identity object (None), Workspace web URL (<https://web.azure-synapse.net/>), and Tags (Click here to add tags). Below this is a 'Getting started' section with a callout box containing the text 'Open Synapse Studio' and 'Start building your fully-integrated analytics solution and unlock new insights.' with a 'Open' button.

Figure 12.2 – Open Synapse Studio

2. Click on the **Develop** button (the notebook-like symbol) on the left, which will take you to the **Develop** section. Click on the + button and select **SQL script**:

The screenshot shows the 'Develop' section of the Synapse live interface. On the left, there is a sidebar with icons for Home, Databases, Tables, and a Develop button (which is highlighted with a red box). The main area shows a list of resources under 'SQL scripts': SQL script 1, SQL script 2, and SQL script 3. A red box highlights the '+ SQL script' button at the top right of the list. To the right of the list, there is a search bar with the placeholder 'Filter resources by name' and a dropdown menu with options: SQL script (highlighted with a red box), KQL script, Notebook, Data flow, and Apache Spark job definition.

Figure 12.3 – SQL script

3. In this recipe, to showcase the impact of data movement, we need to scale up the Synapse dedicated SQL pool to DWU 500. Paste the following script in the new **SQL script** window:

```
ALTER DATABASE packtadesqlpool  
MODIFY (SERVICE_OBJECTIVE = 'DW500c');
```

Select **packtadesqlpool** under **Connect to** and **master** under **Use database**. Then, click on the **Run** button to execute the script:

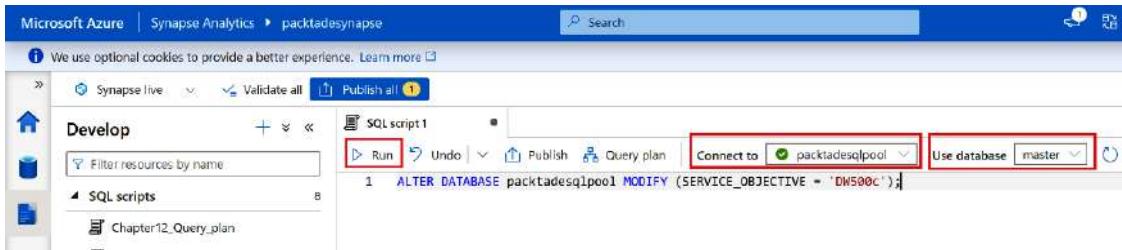


Figure 12.4 – Scaling up

4. Select **packtadesqlpool** under **Use database** and copy and paste the following script in the same query window. The scripts will create two tables called `transaction_tbl_t1` and `transaction_tbl_t2`. Now, `transaction_tbl_t1` is a hash that's been distributed based on the `pid` column, while `transaction_tbl_t2` is a hash that's been distributed based on the `tid` column:

```
CREATE table dbo.transaction_tbl_  
t1 WITH (DISTRIBUTION = HASH(pid))  
AS  
SELECT *  
FROM dbo.transaction_tbl  
  
CREATE table dbo.transaction_tbl_  
t2 WITH (DISTRIBUTION = HASH(tid))  
AS  
SELECT *  
FROM dbo.transaction_tbl
```

Click on the **Run** button to execute the script:

```

1 ALTER DATABASE packtadesqlpool MODIFY (SERVICE_OBJECTIVE = 'DW500c');
2
3 CREATE table dbo.transaction_tbl_t1 WITH (DISTRIBUTION = HASH(pid))
AS
4 SELECT *
5 FROM dbo.transaction_tbl
6
7 CREATE table dbo.transaction_tbl_t2 WITH (DISTRIBUTION = HASH(tid))
AS
8 SELECT *
9 FROM dbo.transaction_tbl
10
11
12

```

Figure 12.5 – Creating a table

- Copy and paste the following script in the same query window. There are two SELECT queries here. The first is an expensive SELECT query, while the second query provides the query plan of the first query. Ensure that you select both queries and run them together as a single batch:

```

Select
t1.sid, AVG(t2.total_cost),max(t1.order_count) as max_
ord_cnt, t2.transaction_date
from dbo.transaction_tbl_t1 t1 inner join dbo.
transaction_tbl_t2 t2
on t1.tid = t2.tid
Group by t1.sid,t2.transaction_date;

Select request_id,operation_type,step_index,row_
count,total_elapsed_time,command from sys.dm_pdw_request_
steps
where request_id in ( Select request_id from sys.dm_pdw_
exec_requests where command like '%Select
t1.sid, AVG(t2.total_cost),max(t1.order_count) as max_%
ord_cnt%'
and session_id in ( select session_id from sys.dm_pdw_
exec_sessions s ) )
and start_time between dateadd(ss,-
20,getdate()) and getdate()
order by total_elapsed_time desc

```

Click on the **Run** button to execute the script:

The screenshot shows the SQL Server Management Studio interface. At the top, there's a toolbar with a 'Run' button highlighted by a red box. Below the toolbar is a code editor window containing a T-SQL script. A large red box highlights the second half of the script, which includes joins between transaction tables and system views like sys.dm_pdw_request_steps and sys.dm_pdw_exec_sessions. The results pane below shows a table with columns: request_id, operation_type, step_index, row_count, total_elapsed_time, and command. The 'total_elapsed_time' column for the 'ShuffleMoveOperation' rows is highlighted with a red box.

request_id	operation_type	step_index	row_count	total_elapsed_t...	command
QID9291	ShuffleMoveOperation	2	242744	609	SELECT [T1_1].[t...]
QID9291	ShuffleMoveOperation	5	118153	328	SELECT [T1_1].[...]
QID9291	OnOperation	7	-1	203	DROP TABLE [q...]
QID9291	ReturnOperation	6	2430	171	SELECT [T1_1].[...]
QID9291	OnOperation	1	-1	109	CREATE TABLE [...]

Figure 12.6 – Getting the query plan

6. The preceding output indicates that **operation_type ShuffleMoveOperation** was the most expensive operation in the query as it had the highest elapsed time of 605 milliseconds and was moving ~242,000 rows across all nodes. **ShuffleMoveOperation** is an internal operation of the Synapse engine, where it attempts to move the data across the nodes to process the query. Click on the **Export results** button and select **CSV** to export the result as a comma-separated file:

21 t1.sid, AVG(t2.total_cost),max(t1.order_count) as max_ord_cnt%
 22 and session_id in (select session_id from sys.dm_pdw_exec_sessions s))
 23 and start_time between dateadd(ss,-20,getdate()) and getdate()
 24 order by total_elapsed_time desc

request_id	operation_type	JSON	row_count	total_elapsed_t...	command
QID9291	ShuffleMoveOperation	2	242744	609	SELECT [T1_1].[t...
QID9291	ShuffleMoveOperation	5	118153	328	SELECT [T1_1].[...
QID9291	OnOperation	7	-1	203	DROP TABLE [ta...

Figure 12.7 – Exporting the query as a CSV file

7. Open the downloaded CSV file in Excel and expand the command column. The first row in the command column contains the internal query used by the engine, which caused a large shuffle movement. Expand the command column; notice that the query selecting the columns required the `transaction_tbl_t1` table. This implies that the engine was shuffling the data of the `transaction_tbl_t1` table across all nodes to process the join condition:

```

  SELECT [T1_1].[tid] AS [tid], [T1_1].[order_count] AS [order_count], [T1_1].[sid] AS [sid] FROM (SELECT [T2_3].[tid] AS [tid], [T2_3].[order_count] AS [order_count], [T2_3].[sid] AS [sid] FROM
  [backtodespool].[dbo].[transaction_tbl_1] AS T2_3 WHERE ((T2_3.[tid] IS NOT NULL) AS T1_1
  OPTION (MAXDOP 4, MIN_GRANT_PERCENT = [MIN_GRANT], DISTRIBUTED_MOVE[N], MAX_GRANT_PERCENT = [MAX_GRANT]))
  
```

request_id	operation_step_index	row_count	total_elaps	command
QID9291	ShuffleMov	2	242744	609 [NULL]) AS
QID9291	ShuffleMov	5	118153	328 SELECT
QID9291	OnOperatik	7	-1	203 DROP TABLE [qtabledb].[dbo].[TEMP_ID_3]

Figure 12.8 – Shuffle operation

8. If we observe the join condition of the expensive SELECT query we submitted in step 5, we will see that the `transaction_tbl_t1` and `transaction_tbl_t2` tables have been joined on the SELECT query using column `tid`. However, in step 4, we created the tables in the following way:

- `transaction_tbl_t1` by hash distributing it using the `pid` column
- `transaction_tbl_t2` by hash distributing it using the `tid` column

A hash distribution stores the data on the nodes of the Synapse dedicated SQL pool based on the value of the column used for hash distribution. In our case, `transaction_tbl_t1` was distributed using the `pid` column, while `transaction_tbl_t2` was distributed based on the `tid` column. To reduce the processing time and the time spent on the shuffle movement operation, the table should be hash distributed using the column that was used for the join operation in the `SELECT` query. As our query has been joined using the `tid` column, let's change the distribution of the `transaction_tbl_t1` table using the `tid` column. Please note that `transaction_tbl_t2` has already been distributed using the `tid` column, so no changes are required. Execute the following script to drop and recreate `transaction_tbl_t1` using the `tid` column:

```
Drop table dbo.transaction_tbl_t1;
CREATE table dbo.transaction_tbl_
t2 WITH (DISTRIBUTION = HASH(tid))
AS
SELECT *
FROM dbo.transaction_tbl
```

Click on the **Run** button to execute the script:



```
SQL script 1 Chapter12_Query_plan
Run Undo Publish Query plan Connect to packtadesqlpool Use database packtadesqlpool
12
13 Select
14 t1.sid, AVG(t2.total_cost),max(t1.order_count) as max_ord_cnt, t2.transaction_date
15 from dbo.transaction_tbl_t1 t1 inner join dbo.transaction_tbl_t2 t2
16 on t1.tid = t2.tid
17 Group by t1.sid,t2.transaction_date;
18
19 Select request_id,operation_type,step_index,row_count,total_elapsed_time,command from sys.dm_pdw_request_steps
20 where request_id in ( Select request_id from sys.dm_pdw_exec_requests where command like '%Select
21 t1.sid, AVG(t2.total_cost),max(t1.order_count) as max_ord_cnt%'
22 and session_id in ( select session_id from sys.dm_pdw_exec_sessions s ) )
23 and start_time between dateadd(ss,-20,getdate()) and getdate()
24 order by total_elapsed_time desc
25
26
27 Drop table dbo.transaction_tbl_t1;
28 CREATE table dbo.transaction_tbl_t1 WITH (DISTRIBUTION = HASH(tid))
29 AS
30 SELECT *
31 FROM dbo.transaction_tbl;
```

Figure 12.9 – Table recreation

9. Rerun the two SELECT queries (the SELECT query joining `transaction_tbl_t1` and `transaction_tbl_t2` and the SELECT query fetching the query plan) we ran in step 5. You will notice that the major **ShuffleMoveOperation** that moved ~242,000 rows across all nodes no longer exists in the query plan and that the query processing time has also improved significantly:

```

SQL script 1 Chapter12_Query_plan
Run Undo Publish Query plan Connect to packtadesqlpool Use database packtadesqlpool
12
13 Select t1.sid, AVG(t2.total_cost),max(t1.order_count) as max_ord_cnt, t2.transaction_date
14 from dbo.transaction_tbl_t1 t1 inner join dbo.transaction_tbl_t2 t2
15 on t1.tid = t2.tid
16 Group by t1.sid,t2.transaction_date;
17
18
19 Select request_id,operation_type,step_index,row_count,total_elapsed_time,command from sys.dm_pdw_request_steps
20 where request_id in ( Select request_id from sys.dm_pdw_exec_requests where command like '%Select'
21 t1.sid, AVG(t2.total_cost),max(t1.order_count) as max_ord_cnt%
22 and session_id in ( select session_id from sys.dm_pdw_exec_sessions s ) )
23 and start_time between dateadd(ss,-20,getdate()) and getdate()
24 order by total_elapsed_time desc
25
26
27 Drop table dbo.transaction_tbl_t1;
28 CREATE table dbo.transaction_tbl_t1 WITH (DISTRIBUTION = HASH(tid))
29 AS
30 SELECT *
31 FROM dbo.transaction_tbl;
32
Results Messages
Select Query 1 View Table Chart Export results
Search
request_id operation_type step_index row_count total_elapsed_t... command
QID9342 ShuffleMoveOperation 2 118153 375 SELECT [T1_1].[...
QID9342 ReturnOperation 3 2430 140 SELECT [T1_1].[...
QID9342 OnOperation 4 -1 109 DROP TABLE [q...
QID9342 OnOperation 1 -1 109 CREATE TABLE [...]
QID9342 RandomIDOperation 0 -1 0 TEMP_ID_12

```

Figure 12.10 – Query plan after redistribution

10. Execute the following script to scale it down to DW100:

```

ALTER DATABASE packtadesqlpool
MODIFY (SERVICE_OBJECTIVE = 'DW100c');

```

How it works...

Aligning the column used for distributing the table with the column used for the join condition ensures that the query optimizer can perform most of the processing at the individual nodes without having to move all the rows of the table across all nodes. Before changing the distribution of the `transaction_tbl_t1` table, we performed a shuffle operation on the `SELECT` query, which moved 242,722 rows. This is the count of all the rows in `transaction_tbl_t1`. `transaction_tbl_t1` was moved completely across the nodes to make the join work. However, once the `transaction_tbl_t1` table was distributed using the `tid` column, the join condition was fully performed concurrently at each node without us having to move all the rows. Only the subset of rows that were required for processing subsequent operations after the join operation (such as the `GROUP BY` and `ORDER BY` clauses) were moved using the shuffle operation, thereby reducing the processing time. It is important to note that while it is nearly impossible to remove all shuffle operations in large queries, eliminating full table movement using the correct choice of distribution column is critical for ensuring the optimal performance of the query.

Monitoring and rebuilding a replication table cache

Replication tables in Synapse dedicated SQL pool are tables that are copied to all the compute nodes and the control node of a Synapse dedicated SQL pool. So, if a Synapse dedicated SQL pool contains three compute nodes, then four copies of the replicated table are present in the database. A replication table prevents the Synapse engine from broadcasting a table across all compute nodes during query processing. However, the replication table would be effective and would prevent broadcast operations when the replication table cache is in the **Ready** state.

This recipe will teach you how to monitor the state of a replication table cache and rebuild the cache for all the replication tables in your Synapse dedicated SQL pool database.

Getting ready

To get started, log into <https://portal.azure.com> using your Azure credentials.

Create a Synapse Analytics workspace, as explained in the *Provisioning an Azure Synapse Analytics workspace* recipe of *Chapter 8, Processing Data Using Azure Synapse Analytics*.

Complete the *Loading data into a dedicated SQL pool using PolyBase and T-SQL* recipe of *Chapter 10, Building the Serving Layer in Azure Synapse SQL Pool*, to create a dedicated SQL pool named `packtadesqlpool`, an external table named `dbo.ext_transaction_tbl`, and a dedicated SQL pool table named `dbo.transaction_tbl`. Alternatively, to create the `dbo.ext_transaction_tbl` and `dbo.transaction_tbl` tables, you may use the `Create_External_Table.SQL` script available at <https://github.com/PacktPublishing/Azure-Data-Engineering-Cookbook-2nd-edition/tree/main/chapter10>.

How to do it...

Follow these steps to create a replication table, monitor the state of the replication table cache, and rebuild the replication table cache to move it into a **Ready** state:

1. Scale up the **packtadesqlpool** instance to DWU500, as described in *steps 1 to 3* of the *How to do it...* section of the *Analyzing a query plan and fixing table distribution* recipe.
2. Select **packtadesqlpool** under **Use database** and copy and paste the following script in the same query window. This script will create a replicated table called **dbo.supplier**:

```
Create table dbo.supplier WITH ( DISTRIBUTION = REPLICATE)
AS
Select distinct sid
[sid],
'S' + '-' + convert(varchar(2),sid) as supplier_name
FROM [dbo].[transaction_tbl_t1]
```

Click on the **Run** button to create the **dbo.supplier** table:

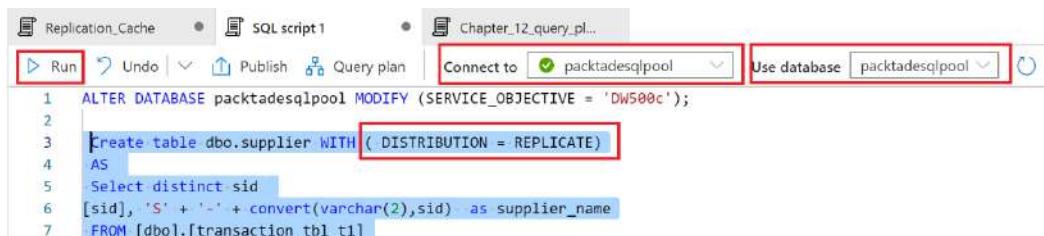


Figure 12.11 – Creating a replicated table

3. Run the following script to check the replication table cache of all replicated tables in any Synapse database. The script queries the **sys.pdw_replicated_table_cache_state Dynamic Management View (DMV)** to find the replication cache's status:

```
SELECT '[' + sch.[name] + '].[' + t.[name] + ']' AS table_name, c.[state] , p.[distribution_policy_desc]
FROM sys.tables t
JOIN sys.pdw_replicated_table_cache_state c
ON c.object_id = t.object_id
JOIN sys.pdw_table_distribution_properties p
```

```

    ON p.object_id = t.object_id
    JOIN sys.schemas sch
        ON t.schema_id = sch.schema_id
    WHERE p.[distribution_policy_desc] = 'REPLICATE'
    ORDER BY c.[state], table_name

```

Click the **Run** button to execute the script and check the replication cache's status. It will be listed as **NotReady**:

The screenshot shows a SQL Server Management Studio (SSMS) interface. The top menu bar includes 'Replication_Cache', 'SQL script 1', 'Chapter_12_query_pl...', 'Run' (button), 'Undo', 'Publish', 'Query plan', 'Connect to', 'packtadesqlpool', 'Use database', 'packtadesqlpool', and a refresh icon. The main area contains a multi-line query editor with the following T-SQL code:

```

3 Create table dbo.supplier WITH ( DISTRIBUTION = REPLICATE)
4 AS
5 Select distinct sid
6 [sid], 'S' + '-' + convert(varchar(2),sid)  as supplier_name
7 FROM [dbo].[transaction_tbl_t1]
8
9 SELECT '[' + sch.[name] + '].[' + t.[name] + '];' AS table_name, c.[state] , p.[distribution_policy_desc]
10   FROM sys.tables t
11  JOIN sys.pdw_replicated_table_cache_state c
12    ON c.object_id=t.object_id
13  JOIN sys.pdw_table_distribution_properties p
14    ON p.object_id=t.object_id
15  JOIN sys.schemas sch
16    ON t.schema_id = sch.schema_id
17 WHERE p.[distribution_policy_desc] = 'REPLICATE'
18 ORDER BY c.[state], table_name
19
20

```

The results pane below shows a table with three columns: 'table_name', 'state', and 'distribution_policy_desc'. The single row returned is:

table_name	state	distribution_policy_desc
[dbo].[supplier]	NotReady	REPLICATE

Figure 12.12 – Checking the replication table's status

4. Copy and paste the following two **SELECT** queries into the query window and execute them together. The first **SELECT** query joins the **dbo.transaction_tbl** and **dbo.supplier** tables and returns some results. The second query prints the execution plan of the first query:

```

Select s.supplier_name,sum(total_cost)
FROM [dbo].[transaction_tbl] t INNER JOIN dbo.supplier s
on t.sid = s.sid

```

```
Group by s.supplier_name
```

```
Select request_id,operation_type,step_index,row_count,
total_elapsed_time,command from sys.dm_pdw_request_steps
where request_id in ( Select request_id from sys.dm_pdw_exec_requests
where command like '%Select s.supplier_name,sum(total_cost)%'
and session_id in ( select session_id from sys.dm_pdw_exec_sessions s )
and start_time between dateadd(ss,-20,getdate()) and getdate()
order by step_index
```

Select both queries and click on the **Run** button to execute them:

request_id	operation_type	step_index	row_count	total_elapsed_time	command
QID9934	RandomIDOperation	0	-1	0	TEMP_ID_35
QID9934	OnOperation	1	-1	31	CREATE TABLE [qtabledb].[d...
QID9934	BroadcastMoveOperation	2	10	62	SELECT [T1_1].[sid] AS [sid], ...
QID9934	RandomIDOperation	3	-1	0	TEMP_ID_36
QID9934	OnOperation	4	-1	125	CREATE TABLE [qtabledb].[d...

Figure 12.13 – Query plan of the replication table in the NotReady state

You will notice that the query plan has a **BroadcastMoveOperation**, which implies that the `dbo.supplier` table was broadcasted across all nodes, even though it was a replicated table. This is the result of the replicated table being in the **NotReady** state.

5. Perform the following steps to rebuild all the replication tables in the **NotReady** state in the Synapse dedicated SQL Pool database:
 - I. Download the `Replication_Cache_Rebuild.SQL` script from https://github.com/PacktPublishing/Azure-Data-Engineering-Cookbook-2nd-edition/blob/main/chapter12/Replication_Cache_Rebuild.sql.
 - II. Click on the **Develop** button (the notebook-like button) on the left. Click on the + symbol and select **Import**:

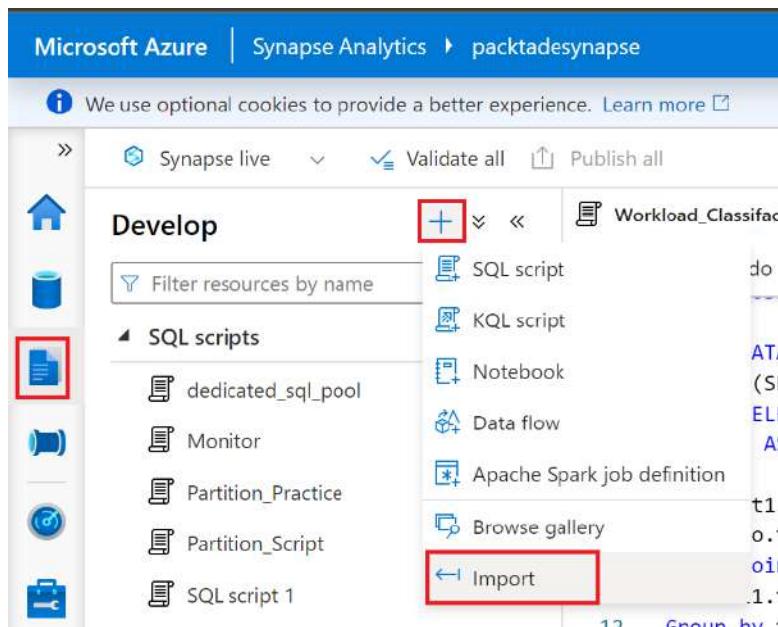
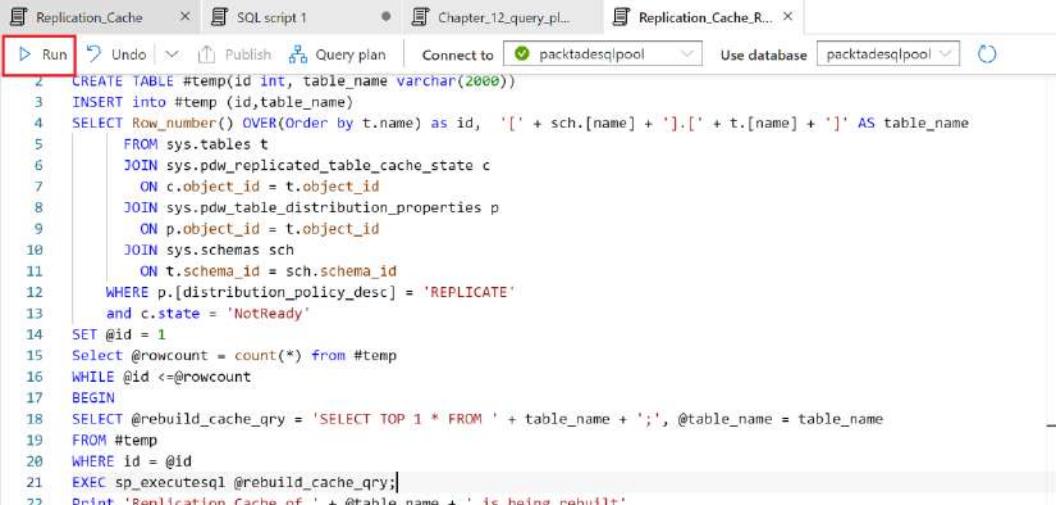


Figure 12.14 – Importing the SQL script

Select the `Replication_Cache_Rebuild.SQL` file you downloaded in *step 1*. Connect to the `packtadesqlpool` database and run the script, as shown in the following screenshot:



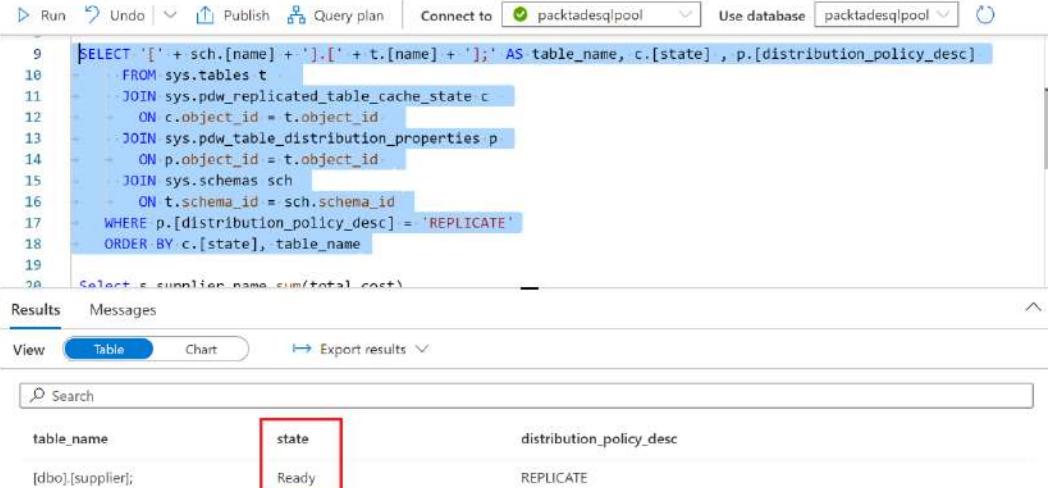
```

2 CREATE TABLE #temp(id int, table_name varchar(2000))
3 INSERT into #temp (id,table_name)
4 SELECT Row_number() OVER(Order by t.name) as id, '[' + sch.[name] + '].[' + t.[name] + ']' AS table_name
5     FROM sys.tables t
6     JOIN sys.pdw_replicated_table_cache_state c
7         ON c.object_id = t.object_id
8     JOIN sys.pdw_table_distribution_properties p
9         ON p.object_id = t.object_id
10    JOIN sys.schemas sch
11        ON t.schema_id = sch.schema_id
12    WHERE p.[distribution_policy_desc] = 'REPLICATE'
13    and c.state = 'NotReady'
14    SET @id = 1
15    Select @rowcount = count(*) from #temp
16    WHILE @id <=@rowcount
17    BEGIN
18        SELECT @rebuild_cache_qry = 'SELECT TOP 1 * FROM ' + table_name + ';', @table_name = table_name
19        FROM #temp
20        WHERE id = @id
21        EXEC sp_executesql @rebuild_cache_qry;
22        Print 'Replication Cache of ' + @table_name + ' is being rebuilt'

```

Figure 12.15 – Rebuilding the replication cache

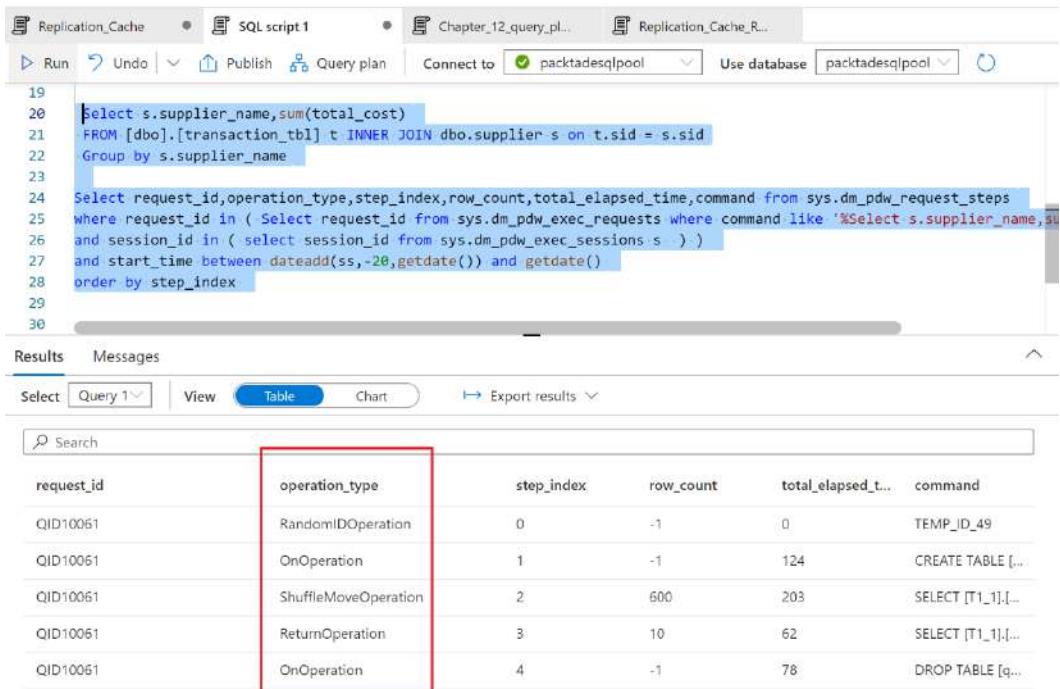
6. Rerun the `SELECT` script in *step 3* that queried the `sys.pdw_replicated_table_cache_state` DMV. Notice that `dbo.supplier` will be in a **Ready** state:



table_name	state	distribution_policy_desc
[dbo].[supplier];	Ready	REPLICATE

Figure 12.16 – The state after rebuilding

7. Rerun the SELECT queries in *step 4* to check whether **BroadcastMoveOperation** has been removed from the query plan:



```

19
20 select s.supplier_name,sum(total_cost)
21 FROM [dbo].[transaction_tbl] t INNER JOIN dbo.supplier s on t.sid = s.sid
22 Group by s.supplier_name
23
24 Select request_id,operation_type,step_index,row_count,total_elapsed_time,command from sys.dm_pdw_request_steps
25 where request_id in ( Select request_id from sys.dm_pdw_exec_requests where command like '%Select s.supplier_name,su
26 and session_id in ( select session_id from sys.dm_pdw_exec_sessions s ) )
27 and start_time between dateadd(ss,-20, getdate()) and getdate()
28 order by step_index
29
30

```

request_id	operation_type	step_index	row_count	total_elapsed_t...	command
QID10061	RandomIDOperation	0	-1	0	TEMP_ID_49
QID10061	OnOperation	1	-1	124	CREATE TABLE [...]
QID10061	ShuffleMoveOperation	2	600	203	SELECT [T1_1];[...]
QID10061	ReturnOperation	3	10	62	SELECT [T1_1];[...]
QID10061	OnOperation	4	-1	78	DROP TABLE [q...]

Figure 12.17 – The query plan after rebuilding

How it works...

The replication table cache moves into the **NotReady** state when an INSERT/UPDATE/DELETE statement or when the table is modified using a **Data Definition Language (DDL)** statement (ALTER/CREATE/Truncate Table statements). The replication table can be moved back to the **Ready** state by issuing a SELECT query against the table. The **Replication_Cache_Rebuild.SQL** script finds the replicated tables in a **NotReady** state and fires a SELECT query against each of them, which rebuilds the replication cache.

Once the table has been rebuilt, a copy of the table is stored in all the compute nodes of the dedicated SQL pool instance, so the query engine doesn't need to broadcast the replicated table to all the nodes during query processing. As a result, the broadcast move operation in the query plan, which was observed when the replicated table was in the **NotReady** state, is not seen once the replicated table is in the **Ready** state.

After loading data into a replicated table, it is always recommended to run the `Select top 1 * from <table name>` command against the table to keep the replication cache ready. You may also schedule the `Replication_Cache_Rebuild.SQL` script to run regularly to keep the replication cache of all replicated tables in the **Ready** state.

Configuring result set caching in Azure Synapse dedicated SQL pool

Result set caching is a feature in Synapse dedicated SQL pools that caches the result of the query in the control node of the Synapse dedicated SQL pool instance. In other words, the first time a query is executed, the result of the query is stored in the control node of the Synapse dedicated SQL pool instance. The next time, when the same query is executed and if the underlying data hasn't changed, the query engine will quickly return the result directly from the cache in the control node without reading the data from any of the compute nodes of the Synapse dedicated SQL pool instance. Unlike other common memory-based caches, the result set cache is persisted even after the Synapse dedicated SQL pool instance has been restarted.

In this recipe, we will learn how to turn on result set caching and how to verify if the cache is being used.

Getting ready

Create a Synapse Analytics workspace, as explained in the *Provisioning an Azure Synapse Analytics workspace* recipe of *Chapter 8, Processing Data Using Azure Synapse Analytics*.

Complete the *Loading data into a dedicated SQL pool using PolyBase and T-SQL* recipe of *Chapter 10, Building the Serving Layer in Azure Synapse SQL Pool*, to create a dedicated SQL pool named `packtadesqlpool`, an external table named `dbo.ext_transaction_tbl`, and a dedicated SQL pool table named `dbo.transaction_tbl`. Alternatively, to create the `dbo.ext_transaction_tbl` and `dbo.transaction_tbl` tables, you may use the `Create_External_Table.SQL` script available at <https://github.com/PacktPublishing/Azure-Data-Engineering-Cookbook-2nd-edition/tree/main/chapter10>.

How to do it...

Follow these steps to configure the result set cache and check whether a query is using the cache:

1. Scale up the `packtadesqlpool` instance to DWU500, as described in *steps 1 to 3* of the *How to do it...* section of the *Analyzing a query plan and fixing table distribution* recipe.

2. Select **packtadesqlpool** under **Use database** and copy and paste the following script in the same query window:

```
Declare @request_id nvarchar(32),@session_id nvarchar(32)
Select
t1.sid, AVG(t2.total_cost),max(t1.order_count) as max_
ord_cnt, t2.transaction_date
from dbo.transaction_tbl t1 inner join dbo.transaction_
tbl t2
on t1.tid = t2.tid
Group by t1.sid,t2.transaction_date;

Select @request_id = req.request_id,@session_id = req.
session_id
from sys.dm_pdw_exec_requests req
where req.command like 'Select
t1.sid, AVG(t2.total_cost),max(t1.order_count) as max_
ord_cnt%'
and req.start_time between dateadd(ss,-
30,getdate()) and getdate()

Select req.request_id, result_cache_hit, req.command,
req.total_elapsed_time as total_query_elapsed_time
from sys.dm_pdw_exec_requests req
where req.request_id = @request_id and req.session_
id = @session_id

Select req_steps.command, req_steps.location_type,req_
steps.step_index,req_steps.operation_type
From sys.dm_pdw_request_steps req_steps
WHERE req_steps.request_id = @request_id
order by req_steps.step_index
```

The result is shown in the following screenshot. Under **Select** on the **Results** pane, select **Query 2**. This query provides the result from the **sys.dm_pdw_request_steps** DMV. Under **location_type**, we can see that the query involved processing and moving the data from a compute node:

command	location_type	step_index	operation_type
TEMP_ID_11	Control	0	RandomIDOperation
CREATE TABLE [qtabledb].[dbo].[...]	Compute	1	OnOperation
SELECT [T1_1].[tid] AS [tid], [T1_1]...	Compute	2	ShuffleMoveOperation
TEMP_ID_12	Control	3	RandomIDOperation
CREATE TABLE [qtabledb].[dbo].[...]	Compute	4	OnOperation
SELECT [T1_1].[tid] AS [tid], [T1_1]...	Compute	5	ShuffleMoveOperation
TEMP_ID_13	Control	6	RandomIDOperation
CREATE TABLE [qtabledb].[dbo].[...]	Compute	7	OnOperation
SELECT [T1_1].[sid] AS [sid], [T1_1]...	Compute	8	ShuffleMoveOperation

00:00:05 Query executed successfully.

Figure 12.18 – Query steps

Select **Query 1** under **Select** on the **Results** pane:

request_id	result_cache_hit	command	total_query_elapsed_time
QID10706	-1	Select t1.sid, AVG(t2.total_cost).m...	1718

Figure 12.19 – Exec_Requests

Query 1 returns the result from **sys.dm_pdw_exec_requests**. The **Result_Cache_hit** column has a value of -1, which indicates that the result set cache was not used. We can also see that the query took **1718** milliseconds to run.

3. In the same query window, select **master** under **Use database** and copy and paste the following script. Click the **Run** button. The script will turn on result set caching:

```
ALTER DATABASE packtadesqlpool SET RESULT_SET_CACHING ON
```

As shown in the following screenshot, the **ALTER DATABASE** command turns on result set caching:

The screenshot shows a SQL Server Management Studio (SSMS) interface. The toolbar at the top includes 'SQL script 1', 'Run' (which is highlighted with a red box), 'Undo', 'Publish', 'Query plan', 'Connect to', and 'Use database'. The 'Use database' dropdown is set to 'master' and is also highlighted with a red box. The main pane contains a multi-line T-SQL script. Line 25 of the script is highlighted with a red box and contains the command: `ALTER DATABASE packtadesqlpool SET RESULT_SET_CACHING ON`. The status bar at the bottom shows 'Results' and 'Messages'.

```
2 Select
3 t1.sid, AVG(t2.total_cost),max(t1.order_count) as max_ord_cnt, t2.transaction_date
4 from dbo.transaction_tbl t1 inner join dbo.transaction_tbl t2
5 on t1.tid = t2.tid
6 Group by t1.sid,t2.transaction_date;
7
8 Select @request_id = req.request_id,@session_id = req.session_id
9 from sys.dm_pdw_exec_requests req
10 where req.command like 'Select'
11 t1.sid, AVG(t2.total_cost),max(t1.order_count) as max_ord_cnt%
12 and req.start_time between dateadd(ss,-30,getdate()) and getdate()
13
14
15 Select req.request_id, result_cache_hit, req.command,
16 req.total_elapsed_time as total_query_elapsed_time
17 from sys.dm_pdw_exec_requests req
18 where req.request_id = @request_id and req.session_id = @session_id
19
20 Select req_steps.command, req_steps.location_type,req_steps.step_index,req_steps.operation_type
21 From sys.dm_pdw_request_steps req_steps
22 WHERE req_steps.request_id = @request_id
23 order by req_steps.step_index
24
25 ALTER DATABASE packtadesqlpool SET RESULT_SET_CACHING ON
```

Figure 12.20 – Turning on result set caching

4. Select **packtadesqlpool** under **Use database** and copy and run the script you executed in *step 2*:

The screenshot shows the Azure Data Studio interface. In the top navigation bar, the 'Use database' dropdown is set to 'packtadesqlpool', which is highlighted with a red box. Below the navigation bar, a T-SQL script is displayed in the 'SQL script 1' tab. The script performs several joins and aggregations on transaction tables to calculate average costs and order counts. In the 'Results' pane, the output of the query is shown in a table format. The table has four columns: 'request_id', 'result_cache_hit', 'command', and 'total_query_elapsed_time'. A single row is present for request ID QID10858, where 'result_cache_hit' is 0, indicating a cache miss.

request_id	result_cache_hit	command	total_query_elapsed_time
QID10858	0	Select t1.sid, AVG(t2.total_cost),max(t1.order_count) as max_ord_cnt, t2.transaction_date from dbo.transaction_tbl t1 inner join dbo.transaction_tbl t2 on t1.tid = t2.tid Group by t1.sid,t2.transaction_date;	9250

Figure 12.21 – Loading the cache

Switch to **Query 1** under **Select** in the **Results** pane. The **result_cache_hit** column has a value of 0, implying that the result set cache was missed. The query didn't use the result set cache as it was fired for the first time since the result set cache was turned. The cache is being loaded at the first run.

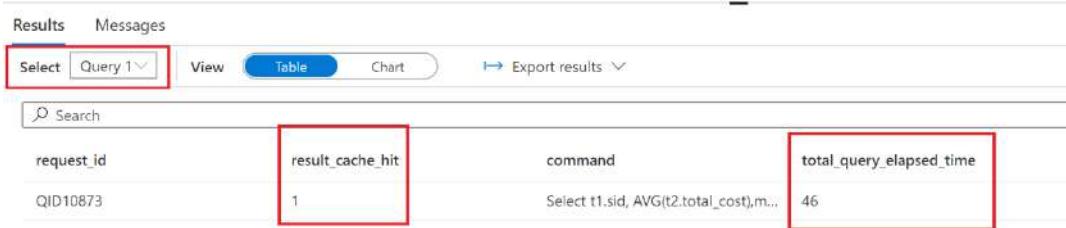
5. Rerun the script you executed in *step 2*. As shown in the **location_type** column of **sys.dm_pdw_request_steps**, the query didn't hit the compute node at all and was fully executed from the control node:



command	location_type	step_index	operation_type
select * from [DWResultCacheDb...	Control	0	ReturnOperation

Figure 12.22 – Control node execution

6. Switch to **Query 1** under **Select** in the **Results** pane. The **result_cache_hit** column has a value of 1, implying that the result set cache was used successfully. The query also ran significantly quickly at **46** milliseconds compared to **1718** milliseconds earlier when result set caching was not turned on:



request_id	result_cache_hit	command	total_query_elapsed_time
QID10873	1	Select t1.sid, AVG(t2.total_cost),m...	46

Figure 12.23 – Cache used successfully

7. Connect to the master database and execute the following script to scale the Synapse dedicated SQL pool down to DW100:

```
ALTER DATABASE packtadesqlpool
MODIFY (SERVICE_OBJECTIVE = 'DW100c') ;
```

How it works...

As we can see, the result set cache significantly reduces the query processing time by caching the results in the control node. However, the result set cache is not used when the underlying table or data changes. When the data changes, the corresponding result set is removed from the cache. The cache is loaded with the updated result set when the first `SELECT` query runs against the updated data.

Hence, result set caching may not be an ideal option for write-heavy environments with massive **Extract, Transform, Load (ETL)** tasks, and in environments with several queries where the resulting size is bigger than 1 GB. In such environments, result set caching may increase the load on the control node as lots of effort by the engine is required to keep the cache updated; this can result in detrimental performance. Result set caching is ideal for environments that have significant reporting workloads with `SELECT` queries making up over 70% of the workload.

There are also other scenarios where the query engine won't use result set caching, even if it is turned on. A few scenarios are when the query uses non-deterministic functions such as `getdate` or when the result set's size is over 10 GB. The list of unsupported scenarios for result set caching can be found at <https://docs.microsoft.com/en-us/azure/synapse-analytics/sql-data-warehouse/performance-tuning-result-set-caching#what's-not-cached>. When a query doesn't use the result set cache, a zero or a negative value is returned by the `sys.dmw_exec_requests` DMV's `result_cache_hit` column. Based on the value returned by the `result_cache_hit` column, we can find the reason why the query missed the result set cache. Information about the `result_cache_hit` column's value and its implication can be found at <https://docs.microsoft.com/en-us/sql/relational-databases/system-dynamic-management-views/sys-dm-pdw-exec-requests-transact-sql?view=aps-pdw-2016-au7#remarks>.

Configuring longer backup retention for a Synapse SQL database

By default, backups of a Synapse SQL dedicated pool database are only available for 7 days. In other words, you can retrieve versions of the database that are no older than 7 days. Having a backup retention period of 7 days in most environments will not be sufficient. Therefore, in this recipe, we will learn how to retain backups with a retention period longer than 7 days.

Getting ready

Create a Synapse Analytics workspace, as explained in the *Provisioning an Azure Synapse Analytics workspace* recipe of *Chapter 8, Processing Data Using Azure Synapse Analytics*.

Create a Synapse dedicated SQL pool named `packtadesqlpool`, as described in steps 1 to 3 of the *How to do it...* section of the *Loading data into a dedicated SQL pool using PolyBase and T-SQL* recipe of *Chapter 10, Building the Serving Layer in Azure Synapse SQL Pool*.

Create an Azure Automation account named `azadeautomation`, as described in steps 3 to 4 of the *How to do it...* section of the *Provisioning and configuring a wake-up script for a serverless SQL database* recipe of *Chapter 5, Configuring and Securing Azure SQL Database*.

How to do it...

In this recipe, we will perform the following tasks to achieve backup retention longer than 7 days:

- Configure an Azure Automation account with the `Az.Accounts` and `Az.Synapse` modules installed and managed identity enabled
- Create a runbook in the Azure Automation account and connect to the Synapse workspace using a managed identity

- Run a PowerShell script from the runbook that does the following:
 - Takes a backup of the Synapse SQL dedicated pool database
 - Creates a new database using the backup created
 - Pauses the newly created database

Follow these steps:

- Log into `portal.azure.com`, go to **All resources**, and search for `azadeautomation`. Look for the **Modules** menu under **Shared resources**. Click on **Browse gallery** and import the following modules in the order specified here:

- `Az.Accounts`
- `Az.Synapse`

While importing, select the runtime version as 7.1 (or above) for both modules. Wait for `Az.Accounts` to be imported before you start importing `Az.Synapse`. For detailed instructions on importing modules, please refer to *steps 5 to 7 of the How to do it... section of the Provisioning and configuring a wake-up script for a serverless SQL database recipe of Chapter 5, Configuring and Securing Azure SQL Database.*

- In the `azadeautomation` account, go to **Identity** under **Account Settings**. Click on the **On** button under **Status** and click **Save**. Turning on **Identity** will enable **managed identity** authentication for your Azure Automation account. Using managed identity, your Azure automation account can connect to the Azure resources without you having to specify your user ID and password. Connecting using the **managed identity** of `azadeautomation` will only work for scripts deployed inside the Azure Automation account, making managed identity a very safe method of authentication:

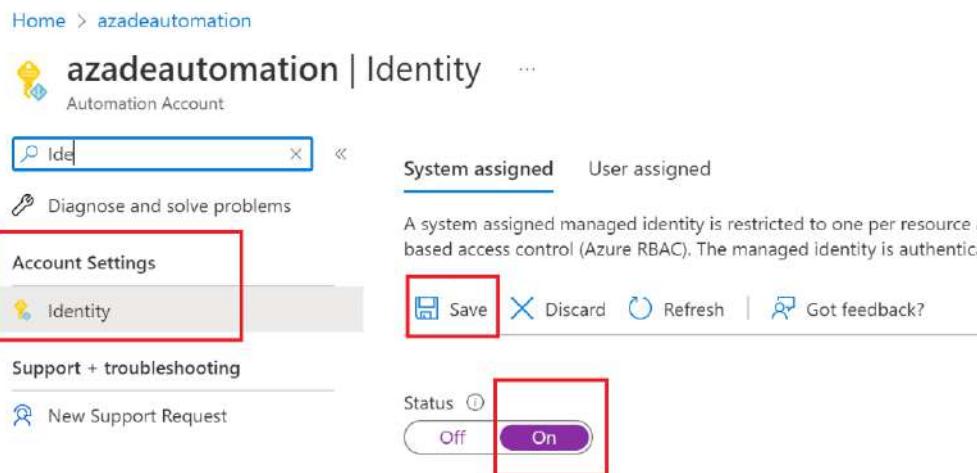


Figure 12.24 – Turning on managed identity

Click **Yes** when prompted for confirmation:



Figure 12.25 – Confirming managed identity creation

3. Click on **Azure role assignments**:

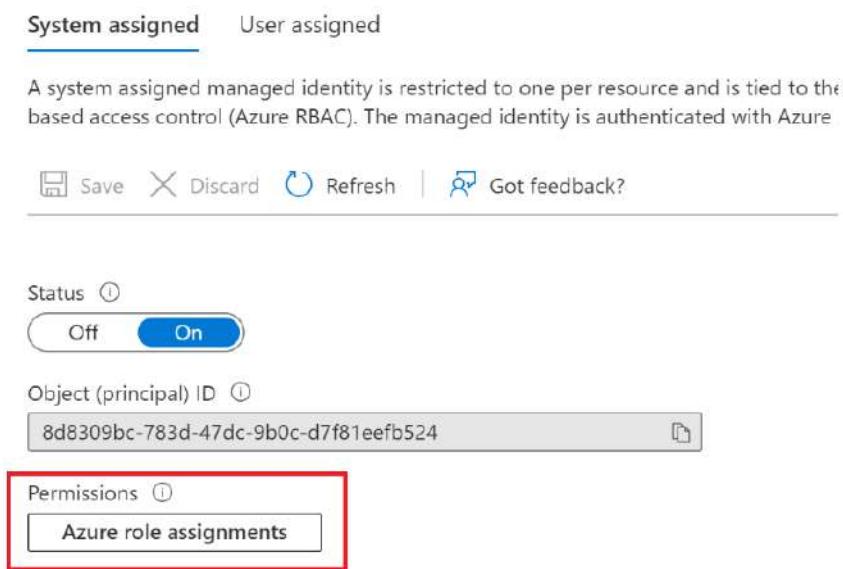


Figure 12.26 – Azure role assignments – managed identity

4. Click on **+ Add role assignment (Preview)**. Set **Scope** to **Subscription** and **Role** to **Reader** and click **Save**:

The screenshot shows the 'Add role assignment (Preview)' interface. It has two main sections: 'Scope' and 'Role'. The 'Scope' section is set to 'Subscription' and lists 'Visual Studio Ultimate with MSDN'. The 'Role' section is set to 'Reader'. At the bottom, there are 'Save' and 'Discard' buttons.

Figure 12.27 – Azure role assignment (Preview) – Subscription and Reader

5. Similarly, add another role assignment. Set **Scope** to **Resource Group**, **Resource Group** to **PacktADESynapse** (the resource group where the Synapse Analytics workspace resides), and **Role** to **Contributor** and click **Save**. Once done, the **Azure role assignments** page should reflect this, as shown in the following screenshot. It can take a few minutes for the roles to be listed. These roles are required for the managed identity of the Azure Automation account to connect to the subscription and back up and restore the Synapse dedicated SQL pool database:

The screenshot shows the 'Azure role assignments' page. It lists two roles assigned to 'azadeautomation': 'Reader' with 'Visual Studio Ultimate with MSDN' as the resource name, and 'Contributor' with 'PacktADESynapse' as the resource name. Both entries show 'Subscription' as the resource type and 'azadeautomation' as the assigned to user.

Role	Resource Name	Resource Type	Assigned To	Condition
Reader	Visual Studio Ultimate with MSDN	Subscription	azadeautomation	None
Contributor	PacktADESynapse	Resource Group	azadeautomation	None

Figure 12.28 – The Azure role assignments page

6. In **azadeautomation**, the Azure Automation account you have created, go to the **Process Automation** section and click on **Runbooks**. Then, click **Create a runbook**:

The screenshot shows the 'azadeautomation | Runbooks' page. At the top, there's a navigation bar with 'Home > azadeautomation'. Below it, the title 'azadeautomation | Runbooks' is displayed next to an 'Automation Account' icon. A search bar labeled 'Run' and a 'Create a runbook' button (highlighted with a red box) are visible. To the right, there are links for 'Import a runbook' and 'B...'. On the left, a sidebar titled 'Process Automation' has a 'Runbooks' item (also highlighted with a red box). The main area shows a table with one record: 'Name: rnscalesql, Authoring status: In edit'. A search bar 'Search runbooks...' is also present.

Figure 12.29 – Create a runbook

7. Set the runbook's **Name** to **SynapseBackup**, **Runbook type** to **PowerShell**, and **Runtime version** to **7.1 (preview)**:

The screenshot shows the 'Create a runbook' form. It has three main input fields: 'Name *' (containing 'SynapseBackup'), 'Runbook type *' (containing 'PowerShell'), and 'Runtime version *' (containing '7.1 (preview)'). Each of these fields is highlighted with a red box. Below the form is a note: 'During runbook execution, PowerShell modules targeting 7.1 runtime version will be used. Please make sure the required PowerShell modules are present in 7.1 runtime version.' At the bottom, there are 'Create' and 'Cancel' buttons, with 'Create' also highlighted with a red box.

Figure 12.30 – The Create a runbook page

8. Copy and paste the following script in the runbook in sequence:

```
#Set Variables
$ResourceGroupName = "PacktADESynapse"
$SynapseAnalyticsWorksapce = "packtadesynapse"
$DatabaseName = "packtadesqlpool"
$label = $DatabaseName + (Get-Date -Format "yyyyMMdd")
```

Assign the appropriate values for the \$ResourceGroupName, \$SynapseAnalyticsWorksapce, and \$DatabaseName variables. The script will create a backup database; the name of the backup database will be the original database name with the date suffixed to it. The \$label variable will contain the new database name.

Copy and paste the following commands into the runbook:

```
#Login using Managed identity
$AzureContext = (Connect-AzAccount -Identity).context
$AzureContext = Set-AzContext -SubscriptionName
$AzureContext.Subscription -DefaultProfile $AzureContext
```

In the preceding code, we have the following:

- Connect-AzAccount -Identity initiates a connection to the Azure subscription using managed identity
- The Set-AzContext command sets the subscription against which the scripts are supposed to run

Copy and paste the following lines of code into the runbook:

```
$pool = Get-AzSynapseSqlPool -ResourceGroupName
$ResourceGroupName -WorkspaceName $SynapseAnalytics
Worksapce -Name $DatabaseName

$databaseId = $pool.Id -replace "Microsoft.Synapse",
"Microsoft.Sql" `

-replace "workspaces", "servers" `

-replace "sqlPools", "databases"

New-AzSynapseSqlPoolRestorePoint -WorkspaceName
$SynapseAnalyticsWorksapce -Name $DatabaseName
-RestorePointLabel $label

# Get the latest restore point
$restorePoint = $pool | Get-AzSynapseSqlPoolRestorePoint
| Select-Object -Last 1

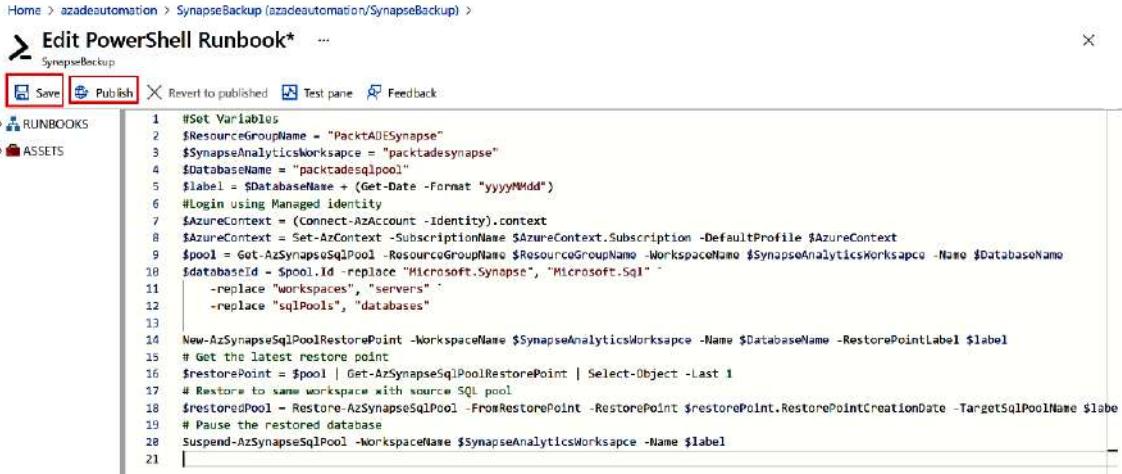
# Restore to same workspace with source SQL pool
```

```
$restoredPool = Restore-AzSynapseSqlPool -FromRestorePoint
-RestorePoint $restorePoint.RestorePointCreationDate
-TargetSqlPoolName $label -ResourceGroupName $pool
.ResourceGroupName -WorkspaceName $pool.WorkspaceName
-ResourceId $databaseId -PerformanceLevel DW100c
# Pause the restored database
Suspend-AzSynapseSqlPool -WorkspaceName
$SynapseAnalyticsWorksapce -Name $label
```

In the preceding code, we have the following:

- The `Get-AzSynapseSqlPool` command gets details about the database that needs to be backed up and restored, assigned to a variable.
- The `New-AzSynapseSqlPoolRestorePoint` command creates a restore point. This restore point will be used to create a backup of the database.
- `Restore-AzSynapseSqlPool` creates a new database using the restore point created. The new database will contain the data as of the restore point and serve as the backup database. For example, if the restore point is created at 10 P.M., the backup database will contain the state of the database as of 10 P.M.
- `Suspend-AzSynapseSqlPool` will pause the new database that was created as the backup database needs to be running if and only if required. Pausing the new backup database keeps the cost of maintaining the backup solution minimal.

Click the **Save** button and then **Publish**:



```
1 #Set Variables
2 $ResourceGroupName = "PacktADESynapse"
3 $SynapseAnalyticsWorksapce = "packtadesynapse"
4 $DatabaseName = "packtadesqlpool"
5 $Label = $DatabaseName + (Get-Date -Format "yyyyMMdd")
6 #Login using Managed identity
7 $AzureContext = (Connect-AzAccount -Identity).context
8 $AzureContext = Set-AzContext -SubscriptionName $AzureContext.Subscription -DefaultProfile $AzureContext
9 $pool = Get-AzSynapseSqlPool -ResourceGroupName $ResourceGroupName -WorkspaceName $SynapseAnalyticsWorksapce -Name $DatabaseName
10 $databaseId = $pool.Id -replace "Microsoft.Synapse", "Microsoft.Sql"
11     -replace "workspaces", "servers"
12     -replace "sqlPools", "databases"
13
14 New-AzSynapseSqlPoolRestorePoint -WorkspaceName $SynapseAnalyticsWorksapce -Name $DatabaseName -RestorePointLabel $label
15 # Get the latest restore point
16 $restorePoint = $pool | Get-AzSynapseSqlPoolRestorePoint | Select-Object -Last 1
17 # Restore to same workspace with source SQL pool
18 $restoredPool = Restore-AzSynapseSqlPool -FromRestorePoint $restorePoint.RestorePointCreationDate -TargetSqlPoolName $label
19 # Pause the restored database
20 Suspend-AzSynapseSqlPool -WorkspaceName $SynapseAnalyticsWorksapce -Name $label
21
```

Figure 12.31 – PowerShell script

9. Click **Start** to start executing the runbook:

The screenshot shows the Azure portal interface for a runbook named 'SynapseBackup'. At the top, there's a search bar and several action buttons: 'Start' (highlighted with a red box), 'View', 'Edit', 'Link to schedule', and 'Add webhook'. Below the header, there are two tabs: 'Overview' (selected) and 'Essentials'. Under 'Overview', there's a summary section with a green checkmark icon, the runbook name, the date and time (7/23/2022, 2:01 PM), and a 'Job' status. Below this are buttons for 'Resume', 'Stop', 'Suspend', and 'Refresh'. The 'Essentials' section contains details about the job, including its ID (6db6b44b-111a-41c7-94ef-c2d69370aa66) and status ('Completed'). It also lists the environment ('Azure') and the user who ran it ('User'). At the bottom, there are tabs for 'Input', 'Output', 'Errors' (which is highlighted with a red box), 'Warnings', 'All Logs', and 'Exception'. The 'Errors' tab shows '0' errors, with a red box highlighting the '0' and an exclamation mark icon. A table below shows log entries, with the first row ('Time') highlighted with a red box and the message 'No errors found for the job' also highlighted with a red box.

Figure 12.32 – Start execution

10. Wait until the runbook's status changes to **Completed**. Ensure no errors are listed:

The screenshot shows the Azure portal interface for the same runbook after it has completed. The top navigation bar includes 'Home', 'azadeautomation', and the runbook name 'SynapseBackup (azadeautomation/SynapseBackup)'. The main content area displays the runbook details: 'Status : Completed' (highlighted with a red box). Below this, it shows the environment ('Azure') and the user ('User'). At the bottom, there are tabs for 'Input', 'Output', 'Errors' (highlighted with a red box), 'Warnings', 'All Logs', and 'Exception'. The 'Errors' tab shows '0' errors, with a red box highlighting the '0' and an exclamation mark icon. A table below shows log entries, with the first row ('Time') highlighted with a red box and the message 'No errors found for the job' also highlighted with a red box.

Figure 12.33 – Run status

11. Go to **All resources** in the Azure portal and search for `packtadesqlpool`. You will find a database called `packtadesqlpool<yyyymmdd>`, as shown in the following screenshot (`<yyyymmdd>` will be replaced with the date when you ran the script):

The screenshot shows the Azure portal's 'All resources' interface. At the top, there's a search bar with the text 'packtadesqlpool' highlighted by a red box. Below the search bar are several filter buttons: 'Subscription equals all', 'Resource group equals all', 'Type equals all', and 'Location equals all'. A 'Unsecure resources' button is visible. The main table lists two items:

	Type	Resource group
<input type="checkbox"/> packtadesqlpool (packtadesynapse/packtadesqlpool)	Dedicated SQL pool	PacktADESynapse
<input type="checkbox"/> packtadesqlpool20220723 (packtadesynapse/packtadesqlpool20220...	Dedicated SQL pool	PacktADESynapse

Figure 12.34 – Backup database result

How it works...

To create more copies of the database, you may schedule the runbook as per your preferred frequency (daily, weekly, and so on). The steps to schedule a runbook were provided in *steps 13 to 18* of the *How to do it...* section of the *Provisioning and configuring a wake-up script for a serverless SQL database* recipe of *Chapter 5, Configuring and Securing Azure SQL Database*. As the backup database is paused by the script, the cost incurred would be the storage cost due to the size of the backup instance. The total additional cost of the solution would depend on the storage cost and the number of copies of the database you retain. The longer the retention period, the higher the cost. You can use the `Remove-AzSynapseSqlPool` PowerShell command to remove older copies of the database that are not required as per your retention policy.

Auto pausing Synapse dedicated SQL pool

As of July 2022, unlike Azure SQL Database and Azure Databricks, the ability to automatically pause the database is not available in Synapse dedicated SQL pool databases. It would be useful to detect inactive periods in a Synapse dedicated SQL pool database and automatically pause it. This recipe will showcase a method that detects whether the Synapse dedicated SQL pool database was inactive for 30 minutes and pauses it if so.

Getting ready

Create a Synapse Analytics workspace, as explained in the *Provisioning an Azure Synapse Analytics workspace* recipe of *Chapter 8, Processing Data Using Azure Synapse Analytics*.

Create a Synapse SQL dedicated pool database named **packtadesqlpool**, as described in *steps 2 to 3* of the *How to do it...* section of the *Loading data into dedicated SQL pool using PolyBase and T-SQL* recipe of *Chapter 10, Building the Serving Layer in Azure Synapse SQL Pool*.

Create an Azure Automation account named **azadeautomation**, as described in *steps 3 and 4* of the *How to do it...* section of the *Provisioning and configuring a wake-up script for a serverless SQL database* recipe of *Chapter 5, Configuring and Securing Azure SQL Database*.

How to do it...

In this recipe, we will perform the following tasks to pause the Synapse dedicated SQL pool database when it is inactive for 30 minutes:

- Configure an Azure Automation account with the `SqlServer`, `Az.Accounts`, and `Az.Synapse` modules installed and managed identity enabled
- Add a credential to the Azure Automation account to connect to a dedicated SQL pool database
- Create a runbook in the Azure Automation account and execute a PowerShell script from the runbook that performs the following tasks:
 - Checks, using Synapse DMVs, whether there have been no active transactions for the last 30 minutes
 - Pauses the database if the database is inactive

Follow these steps:

1. Log into `portal.azure.com`, go to **All resources**, and search for **azadeautomation**. Look for the **Modules** menu under **Shared resources**. Click on **Browse gallery** and import the following modules in the order specified here:
 - `Az.Accounts`
 - `Az.Synapse`
 - `SqlServer`

While importing, select the runtime version as 7.1 (or above) for both modules. Ensure that `Az.Accounts` has been imported before importing `Az.Synapse`. For detailed instructions on importing modules, please refer to *steps 5 to 7* of the *How to do it...* section of the *Provisioning and configuring a wake-up script for a serverless SQL database* recipe of *Chapter 5, Configuring and Securing Azure SQL Database*.

2. In the **azadeautomation** account, perform the following tasks:
 - Enable a system-managed identity account in the **Identity** section
 - Grant the following permissions:
 - **Reader** for **Azure Subscription**
 - **Contributor** for the **PacktADESynapse** resource group
 - Create an empty runbook called **SynapseAutoPause** on version 7.1
- For detailed screenshots of the preceding tasks, please refer to *steps 2 to 7* of the *How to do it...* section of the *Configuring longer backup retention for a Synapse SQL database* recipe.
3. Go to **Credentials** in the **Shared resources** section of the **azadeautomation** account and create a credential named **SynapseCred**. Specify the username as **sqladminuser** and the password as **PacktAdeSynapse123**. For detailed instructions and screenshots on a similar task, please refer to *steps 8 and 9* of the *How to do it...* section of the *Provisioning and configuring a wake-up script for a serverless SQL database* recipe of *Chapter 5, Configuring and Securing Azure SQL Database*.

This credential contains the SQL database user ID and password to be used to connect to the Synapse dedicated SQL pool database; these were specified while creating the Azure Synapse Analytics workspace.

4. Paste the following script:

```
$ResourceGroupName = "PacktADESynapse"
$SynapseAnalyticsWorksapce = "packtadesynapse"
$DatabaseName = "packtadesqlpool"
$instanceName = $SynapseAnalyticsWorksapce + ".sql.
azuresynapse.net"
```

Assign the appropriate values for the `$ResourceGroupName`, `$SynapseAnalyticsWorksapce`, and `$DatabaseName` variables.

Copy and paste the following code into the runbook:

```
#Login using Managed identity
$AzureContext = (Connect-AzAccount -Identity).context
$AzureContext = Set-AzContext -SubscriptionName
$AzureContext.Subscription -DefaultProfile $AzureContext
```

In the preceding script, we have the following:

- `Connect-AzAccount -Identity` initiates a connection to the Azure subscription using managed identity

- The `Set-AzContext` command sets the subscription against which the scripts are supposed to run

Copy and paste the following script into the runbook:

```
$Query = "
    select count(*) as request_count from sys.dm_pdw_exec_
    requests req inner join sys.dm_pdw_exec_sessions ss on
    ss.session_id = req.session_id
    where
        (req.status in ('Running','Suspended') or (
        req.submit_time > DATEADD(minute, -30, GETDATE()) or
        req.start_time > DATEADD(minute, -30, GETDATE()) or req.
        end_time >
        DATEADD(minute, -30, GETDATE())))
        and req.[label] not like 'SynapseAutoPause Job' and ss.
        app_name not in ('Internal') OPTION (LABEL =
        'SynapseAutoPause Job')"
```

Here, we are creating a variable called `$Query` that contains the query based on the `sys.dm_pdw_exec_requests` and `sys.dm_pdw_exec_sessions` DMVs to count the number of active queries/requests in the last 30 minutes. The script takes into account any query that was running, started, or finished within the last 30 minutes. The script excludes system sessions and any connection from the `SynapseAutoPause` runbook. If the count of requests is equal to zero, then the database is said to be inactive and it qualifies to be paused.

Copy and paste the following code into the runbook:

```
$pool = Get-AzSynapseSqlPool -ResourceGroupName
$ResourceGroupName -WorkspaceName
$SynapseAnalyticsWorksapce -Name $DatabaseName

if ($pool.Status -like 'paused' )
{
    Write-Output "Synapse SQL DB is already paused"
}
```

The `Get-AzSynapseSqlPool` command gets the current status of the Synapse dedicated SQL pool database. If the database is already paused, no action is needed from the script and it prints a message stating **Synapse SQL DB is already paused**.

Copy and paste the following code into the runbook:

```
else
{
    $result = invoke-sqlcmd
```

```

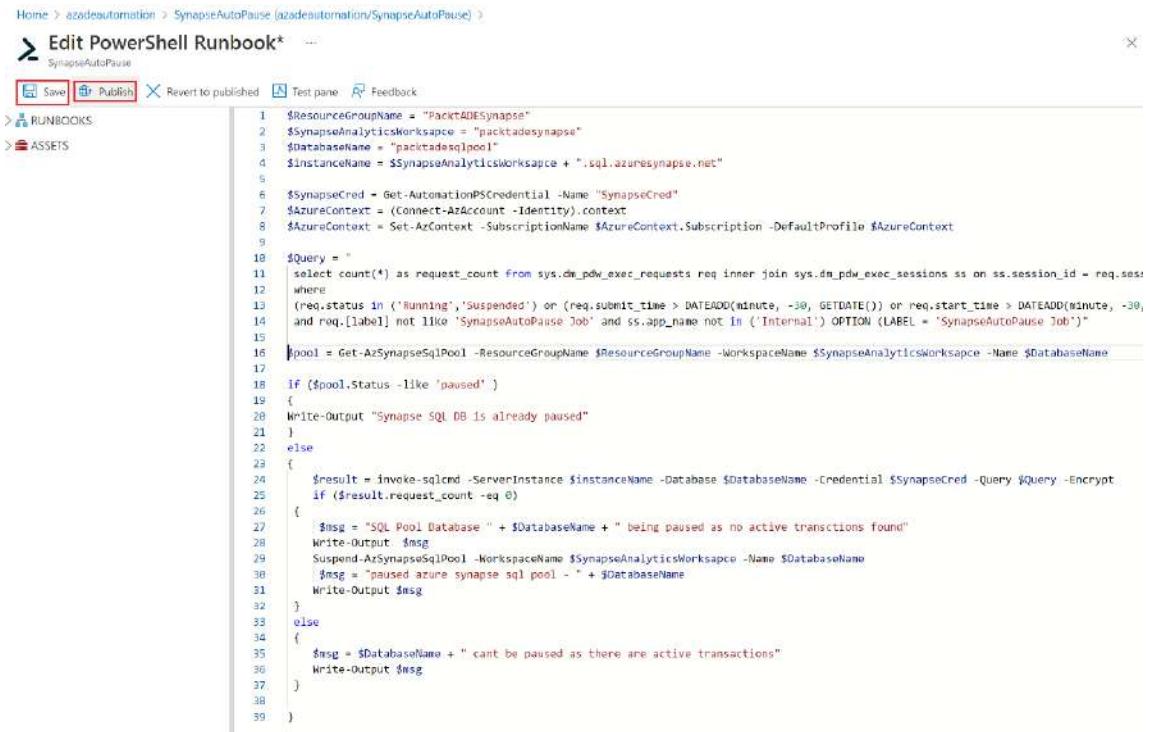
-ServerInstance $instanceName
-Database $DatabaseName -Credential
$SynapseCred -Query $Query -Encrypt
    if ($result.request_count -eq 0)
    {
        $msg = "SQL Pool Database " +
"$DatabaseName + " being paused as no
active transactions found"
        Write-Output $msg
        Suspend-AzSynapseSqlPool -WorkspaceName
$SynapseAnalyticsWorksapce -Name $DatabaseName
        $msg = "paused azure synapse sql pool
- " + $DatabaseName
        Write-Output $msg
    }
    else
    {
        $msg = $DatabaseName + " cant be paused
as there are active transactions"
        Write-Output $msg
    }
}

```

In the preceding code, we have the following:

- The `invoke-sqlcmd` command executes the query we prepared to check whether the database is inactive. The result of the query is assigned to a variable named `$result`.
- The `if ($result.request_count -eq 0)` condition checks if 0 user queries were running in the last 30 minutes to establish whether the database was inactive. If the condition is found to be true, the `Suspend-AzSynapseSqlPool` command is fired to pause the database.
- If the `if ($result.request_count -eq 0)` condition is evaluated to false, it implies there is at least one active query from any user in the last 30 minutes, and the script will exit without pausing the database.

Click the **Save** button and then **Publish**:



```

1 $ResourceGroupName = "PacktADESynapse"
2 $SynapseAnalyticsWorkspace = "packtadesynapse"
3 $DatabaseName = "packtadessqlpool"
4 $InstanceName = $SynapseAnalyticsWorkspace + ".sql.azuresynapse.net"
5
6 $SynapseCred = Get-AutomationPSCredential -Name "SynapseCred"
7 $AzureContext = (Connect-AzAccount -Identity).context
8 $AzureContext = Set-AzContext -SubscriptionName $AzureContext.Subscription -DefaultProfile $AzureContext
9
10 $Query = "
11 select count(*) as request_count from sys.dm_pdw_exec_requests req inner join sys.dm_pdw_exec_sessions ss on ss.session_id = req.session_id
12 where
13 (req.status in ('Running','Suspended') or (req.submit_time > DATEADD(minute, -30, GETDATE()) or req.start_time > DATEADD(minute, -30,
14 and req.[label] not like 'SynapseAutoPause Job' and ss.app_name not in ('Internal')) OPTION (LABEL = 'SynapseAutoPause Job'))
15
16 $pool = Get-AzSynapseSqlPool -ResourceGroupName $ResourceGroupName -WorkspaceName $SynapseAnalyticsWorkspace -Name $DatabaseName
17
18 IF ($pool.Status -like 'paused')
19 {
20 Write-Output "Synapse SQL DB is already paused"
21 }
22 else
23 {
24 $result = invoke-sqlcmd -ServerInstance $instanceName -Database $DatabaseName -Credential $SynapseCred -Query $Query -Encrypt
25 if ($result.request_count -eq 0)
26 {
27 $msg = "SQL Pool Database " + $DatabaseName + " being paused as no active transactions found"
28 Write-Output $msg
29 Suspend-AzSynapseSqlPool -WorkspaceName $SynapseAnalyticsWorkspace -Name $DatabaseName
30 $msg = "Paused azure synapse sql pool - " + $DatabaseName
31 Write-Output $msg
32 }
33 else
34 {
35 $msg = $DatabaseName + " can't be paused as there are active transactions"
36 Write-Output $msg
37 }
38 }
39 }

```

Figure 12.35 – Publishing the auto pause script

5. Let's test the script. In the Azure portal, go to **All resources** and search for **PacktADESynapse**, the Synapse Analytics workspace. Open **Synapse Studio**. Click on the notebook-like icon on the left-hand side of the screen. This will take you to the **Develop** section. Click the **+** button at the top and click on **SQL script**:

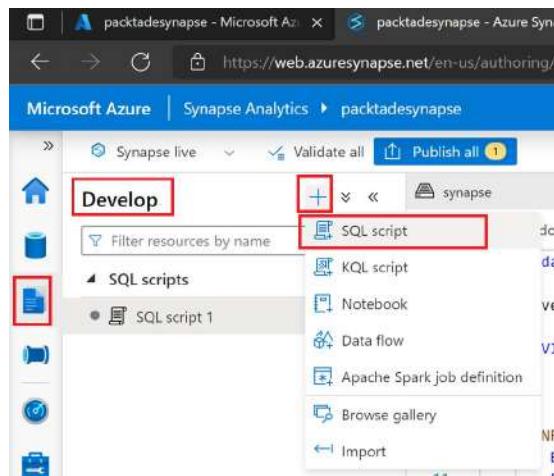


Figure 12.36 – New SQL script

6. Select **packtadesqlpool** under **Connect to**. Insert a simple query such as **Select GETDATE()**, as shown in the following screenshot, and click **Run**:

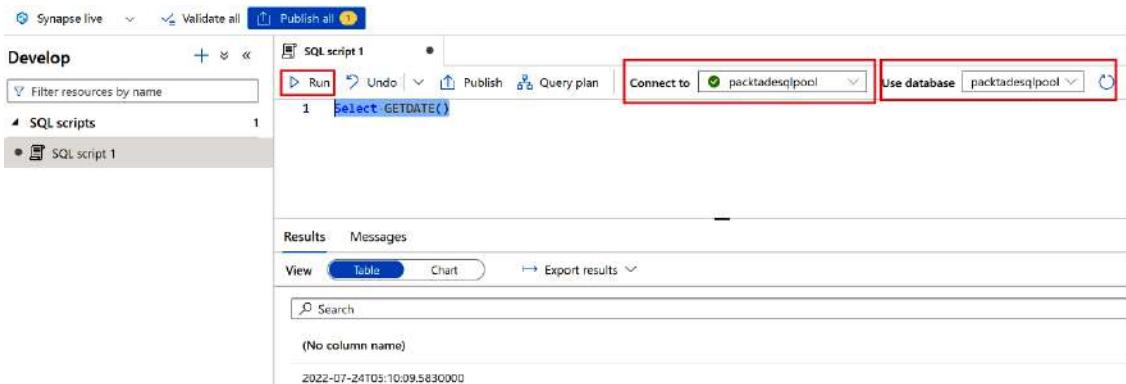


Figure 12.37 – Running the SQL script

7. Don't run any other query on the **packtadesqlpool** database. Wait for 30 minutes or longer and go back to the **azadeautomation** account in the Azure portal. Then, click on **Runbooks** under **Process Automation** and click the **SynapseAutoPause** runbook:

Name	Authoring status	Runbook type	Runtime version	Last modified
mssqlsql	⚠ In edit	PowerShell Workflow	5.1	1/22/2022, 8:02 PM
SynapseAutoPause	✓ Published	PowerShell	7.1 (preview)	7/24/2022, 1:15 PM

Figure 12.38 – Opening the runbook

8. Click on the **Start** button to start the runbook. Click **Yes** when prompted for confirmation to start the runbook:

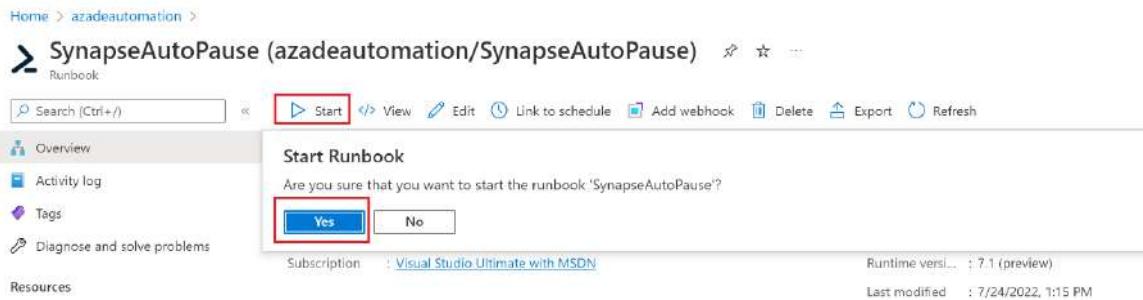


Figure 12.39 – Starting the runbook

9. Verify that the runbook has finished running without any errors:

The screenshot shows the Azure portal interface for a job named 'SynapseAutoPause' from 7/24/2022, 2:37 PM. The top navigation bar includes 'Home > azadeautomation > SynapseAutoPause (azadeautomation/SynapseAutoPause) >'. The main title is 'SynapseAutoPause 7/24/2022, 2:37 PM'. Below the title are buttons: 'Resume' (highlighted with a red box), 'Stop', 'Suspend', and 'Refresh'. The 'Job' section has an 'Essentials' panel with the following details:

- Id : ba29b680-c9aa-4de6-9372-9be60c05c187
- Status : Completed (highlighted with a red box)
- Ran ... : Azure
- Ran ... : User

On the right, there are details: Created : 7/24/2022, 2:37:10 PM, Last Update : 7/24/2022, 2:37:42 PM, Runbook : SynapseAutoPause, Source snaps... : View source snapshot. Below this is a navigation bar with tabs: Input, Output, Errors (highlighted with a red box), Warnings, All Logs, and Exception. Under 'Errors', it says 0 !. A table below shows columns: Time, Type, and Details, with the note 'No errors found for the job'.

Figure 12.40 – Starting the runbook

10. Go to **All resources** in the Azure portal and search for `packtadesqlpool`, the Synapse dedicated SQL pool database. Click on it. Observe its status and notice that the database's status is **Paused**. The database was paused by the runbook as there was no activity for 30 minutes:

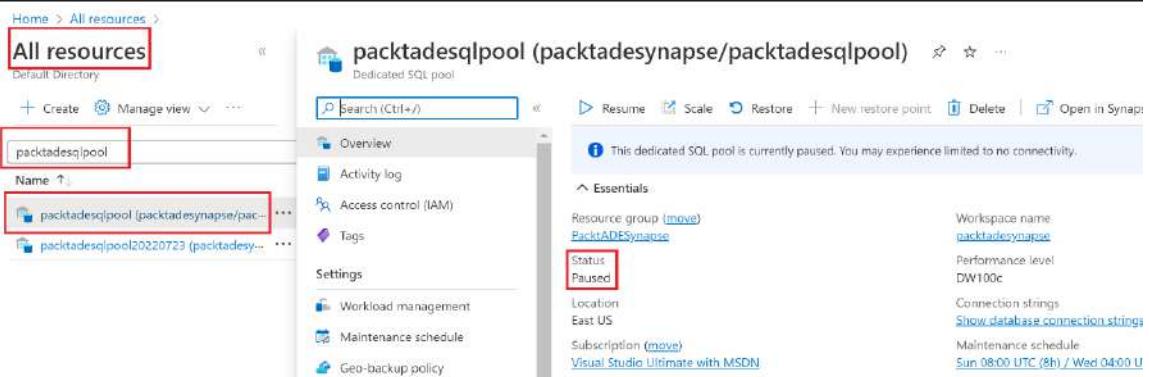


Figure 12.41 – Paused database

How it works...

In this recipe, we created a runbook called **SynapseAutoPause** in our Azure Automation account to run a PowerShell script that pauses the Synapse dedicated SQL pool database if the database is inactive for 30 minutes. To test the runbook, we ran a simple SQL script against the Synapse dedicated SQL pool database and left the database quiet for 30 minutes. After 30 minutes, we ran the runbook and verified that the database has been paused. The runbook can be scheduled at a suitable frequency to check if the database is inactive and pause it to reduce the cost incurred from the Synapse dedicated SQL pool. The steps to schedule a runbook were provided in *steps 13 to 18* of the *How to do it...* section of the *Provisioning and configuring a wake-up script for a serverless SQL database* recipe of *Chapter 5, Configuring and Securing Azure SQL Database*. The script that was used in this recipe is suitable for development, **user acceptance testing (UAT)** environments, and projects where the Synapse dedicated SQL pool may not need to be running all the time.

Optimizing Delta tables in a Synapse Spark pool lake database

As covered in the *Processing data using Spark pools and lake databases* recipe of *Chapter 8, Processing Data Using Azure Synapse Analytics*, a lake database allows you to store processed data in Delta tables, which are powered by Parquet files. Delta tables are very suitable for storing processed data that can be consumed by reporting solutions such as Power BI.

To achieve optimal performance in Delta tables, it is essential to evenly distribute the data among the Parquet files and purge the unwanted ones. The `OPTIMIZE` command helps optimally distribute the data among Parquet files, while the `VACUUM` command purges redundant Parquet files from the Azure Data Lake filesystem. The `OPTIMIZE` and `VACUUM` commands need to be executed regularly on the lake database so that you have optimal performance for the queries run against Delta tables.

In this recipe, we will be writing a script that can scan all Delta tables, optimize them, and vacuum redundant Parquet files.

Getting ready

To get started, log into `https://portal.azure.com` using your Azure credentials.

Create a Synapse Analytics workspace, as explained in the *Provisioning an Azure Synapse Analytics workspace* recipe of *Chapter 8, Processing Data Using Azure Synapse Analytics*.

Create a Spark pool cluster, as explained in the *Provisioning and configuring Spark pools* recipe of *Chapter 8, Processing Data Using Azure Synapse Analytics*.

Download the `transaction-tbl.csv` file from `https://github.com/PacktPublishing/Azure-Data-Engineering-Cookbook-2nd-edition/tree/main/chapter10`. In the Synapse Analytics workspace, create a folder named `files` in the data lake account attached to it. Upload the `transaction-tbl.csv` file to the `files` folder. For detailed screenshots of a similar task, follow steps 1 to 4 in the *How to do it...* section of the *Analyzing data using a serverless SQL pool* recipe of *Chapter 8, Processing Data Using Azure Synapse Analytics*.

How to do it...

Follow these steps to run a script that can optimize and vacuum all delta tables:

1. First, let's create a lake database and a few Delta tables. Download the `Create_Delta_Table.ipynb` notebook from `https://github.com/PacktPublishing/Azure-Data-Engineering-Cookbook-2nd-edition/blob/main/chapter12`.
2. In **Synapse Studio**, click on the **Develop** button (the notebook-like button) on the left. Click on the **+** symbol, select **Import**, and select the `Create_Delta_Table.ipynb` file that we downloaded:

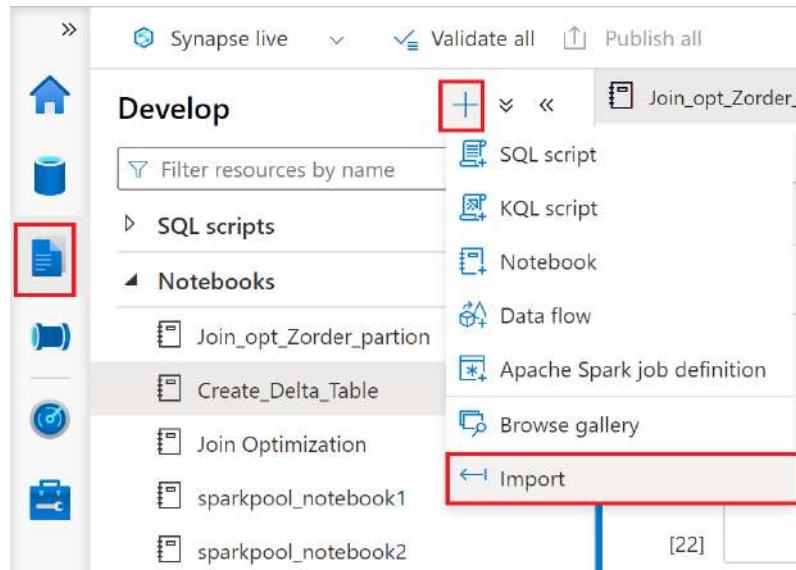


Figure 12.42 – Import

3. Go to the **Create_Delta_Table** notebook. Select **packtsparkpool** under **Attach to** and click **Run all**. The notebook will create a lake database called **lake_db** and two tables called **lake_db.transaction_tbl_t1** and **lake_db.transaction_tbl_t2**. Once the tables have been created, the notebook will perform three **INSERT** statements in the **lake_db.transaction_tbl_t1** table:

The screenshot shows the Microsoft Azure Synapse Analytics portal with the 'Create_Delta_Table' notebook open. The 'Attach to' dropdown is set to 'packtsparkpool' (highlighted with a red box). The 'Run all' button is also highlighted with a red box. The notebook code is visible in the editor, and the output pane shows the results of the job execution, indicating success. A preview of the data in the 'transaction_tbl_t1' table is shown at the bottom.

```

1 %spark
2 df = spark.read.load('abfss://synapse@packtadesynapse.dfs.core.windows.net/files/transaction-tbl.csv', for
3 ## If header exists uncomment line below
4 ,header=True
5 )
6 display(df.limit(10))
    
```

Figure 12.43 – Creating a Delta table

4. Click on the + symbol and create a new notebook:

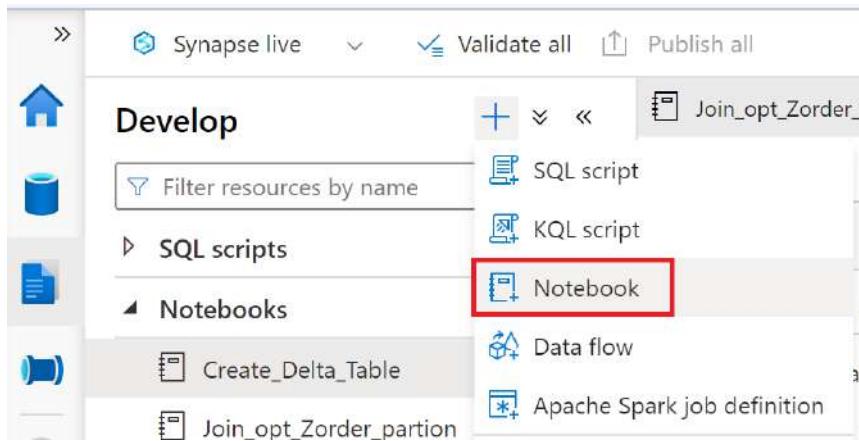


Figure 12.44 – Creating a new notebook

5. Select **packtsparkpool** under **Attach to**. Copy the following script and click the **Run** (small triangle) button:

```
%%sql
Describe detail lake_db.transaction_tbl_t1;
Describe detail lake_db.transaction_tbl_t2;
```

Scroll to the right and observe the **numfiles** and **sizeinBytes** columns. Notice that although the **lake_db.transaction_tbl_t1** and **lake_db.transaction_tbl_t2** tables are similar in size, the **lake_db.transaction_tbl_t1** table contains 18 files as we ran some additional insert queries to insert a few rows:

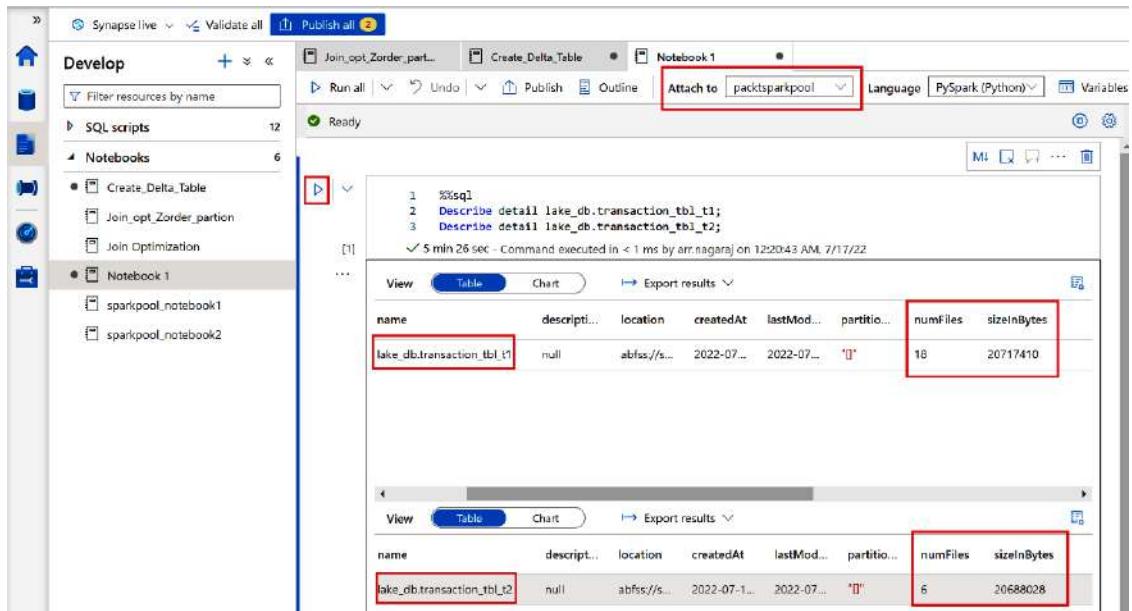


Figure 12.45 – Number of files

- Add a new cell by hovering your mouse below the result set area of the previous cell and clicking the + **Code** button. Copy and paste the following script and click the **Run** button. This script runs the VACUUM and OPTIMIZE commands for all the tables in the database:

```

try:
    database_name = "lake_db"
    tables = spark.sql(f"SHOW TABLES FROM `{{database_name}}`").select("tableName").collect()
    tables = [(row.tableName) for row in tables]
    for table_name in tables:
        spark.sql(f"OPTIMIZE `{{database_name}}`.`{{table_name}}`")
        spark.sql(f"VACUUM `{{database_name}}`.`{{table_name}}` RETAIN 720 HOURS")
    except Exception as ex:
        raise Exception(f"error:{str(ex)}")

```

The script loops through all the tables in the **lake_db** database and executes the OPTIMIZE and VACUUM commands. By default, Delta tables retain previous versions of the tables for up to 30 days, which is useful for audit purposes. Executing the VACUUM command deletes previous versions of the table; older version data may not be retrievable after the VACUUM command is executed. To prevent loss of older version data, you can add the RETAIN 720 HOURS option after the VACUUM `^{\{database_name\}}.^{\{table_name\}}` command. The RETAIN 720 HOURS option will ensure only the versions that are older than 30 days from the current version of the table are deleted. You may customize the retention hours based on your organization's data retention requirements:

The screenshot shows a Jupyter Notebook interface. A red box highlights the '+ Code' button in the toolbar. The code cell contains the following Python script:

```
1 try:
2     database_name = "lake_db"
3     tables = spark.sql(f"SHOW TABLES FROM `'{database_name}'`").select("tableName").collect()
4     tables = [(row.tableName) for row in tables]
5     for table_name in tables:
6         spark.sql(f"OPTIMIZE `'{database_name}`.`{table_name}`")
7         spark.sql(f"VACUUM `'{database_name}`.`{table_name}` RETAIN 720 HOURS")
8     except Exception as ex:
9         raise Exception(f"error: {str(ex)}")
10
11
```

[9] ✓ 24 sec - Command executed in 22 sec 985 ms by arr.nagaraj on 12:49:30 AM, 7/17/22

Job execution Succeeded Spark 2 executors 8 cores View in monitoring Open Spark history

...

Figure 12.46 – Vacuuming and optimizing all tables

7. Rerun the `Describe detail <database_name>. <table_name>` script that you ran in step 5. Notice that both tables now only have one Parquet file since the OPTIMIZE command has optimized the data distribution:

● Session timed out. Run the notebook to start a new session.

name	descripti...	location	createdAt	lastMod...	partitio...	numFiles	sizeInBytes	prope
lake_db.transaction_tbl_t1	null	abfss://s...	2022-07...	2022-07...	"[]"	1	20082987	"{}"

name	descripti...	location	createdAt	lastMod...	partitio...	numFiles	sizeInBytes	
lake_db.transaction_tbl_t2	null	abfss://s...	2022-07...	2022-07...	"[]"	1	20085253	

Figure 12.47 – Post-optimization

How it works...

VACUUM and OPTIMIZE are fundamental commands for maintaining the health of Delta tables. The script uses a simple `for` loop to iterate through all the tables and perform the OPTIMIZE and VACUUM operations. The script can easily be saved as a notebook and scheduled via integration pipelines to regularly optimize the Delta tables and maintain them in a good state.

Optimizing query performance in Synapse Spark pools

There are several methods you can use to optimize the performance of queries in a lake database, such as caching, indexing, partitioning, Z-ordering, data skipping, and using query hints. This recipe will showcase the following two methods to optimize the performance of a query:

- **Z-ordering:** Z-ordering helps the Spark engine easily locate columns with the same value
- **Partitioning:** Partitioning will partition the Delta lake table into smaller chunks, creating subfolders in the data lake storage account for each distinct value on the partitioned column

Getting ready

To get started, log into <https://portal.azure.com> using your Azure credentials.

Create a Synapse Analytics workspace, as explained in the *Provisioning an Azure Synapse Analytics workspace* recipe of *Chapter 8, Processing Data Using Azure Synapse Analytics*.

Create a Spark pool cluster, as explained in the *Provisioning and configuring Spark pools* recipe of *Chapter 8, Processing Data Using Azure Synapse Analytics*.

Download the `transaction-tbl.csv` file from <https://github.com/PacktPublishing/Azure-Data-Engineering-Cookbook-2nd-edition/tree/main/chapter10>. In the Synapse Analytics workspace, create a folder named `files` in the data lake account attached to it. Upload the `transaction-tbl.csv` file to the `files` folder. For detailed screenshots for a similar task, follow steps 1 to 4 in the *How to do it...* section of the *Analyze data using a serverless SQL pool* recipe of *Chapter 8, Processing Data Using Azure Synapse Analytics*.

How to do it...

Follow these steps to optimize a query for reading Delta tables:

1. Download the `Optimize_Delta_Questions.ipynb` notebook from <https://github.com/PacktPublishing/Azure-Data-Engineering-Cookbook-2nd-edition/blob/main/chapter12>. Import the notebook into Synapse Studio, as we did in steps 1 and 2 in the *How to do it...* section of the *Optimizing Delta tables in a Synapse Spark Pool lake database* recipe.
2. Go to the `Optimize_Delta_Questions` notebook. Select `packtsparkpool` under **Attach to** and click **Run all**:

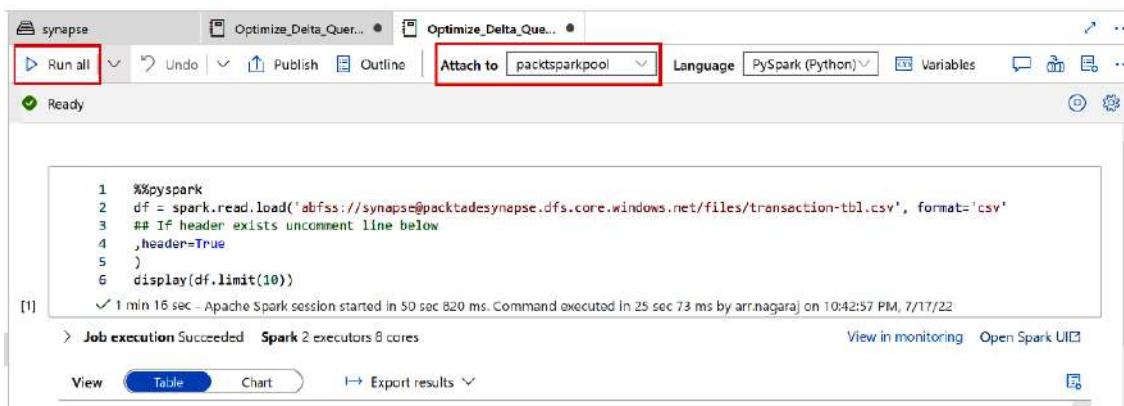


Figure 12.48 – Importing the notebook

3. The first six cells do the following:

- Read the file and create two tables called `transaction_tbl_f1` `transaction_tbl_f2`:
 - `transaction_tbl_f1` contains approximately 25 million rows
 - `transaction_tbl_f2` contains approximately 500,000 rows

4. Go to the last cell in the notebook. This cell runs a SQL script that does the following:

- Joins the `transaction_tbl_f1` and `transaction_tbl_f2` tables on the `tid` column
- Filters by the `pid` column
- Groups the result by the `tid` column

The query joins the mid-sized table, `transaction_tbl_f2`, which contains 500,000 rows with the large table, `transaction_tbl_f1`, which contains 25 million rows. The query takes 37 seconds to run. In this recipe, we will explore options to optimize this query:

+ Code + Markdown

```
1 %%sql
2 Select t1.pid,sum(t2.order_count)
3 FROM transaction_tbl_f1 t1
4 inner join transaction_tbl_f2 t2 on t1.tid = t2.tid
5 WHERE t1.pid between 3 and 7
6 Group by t1.pid
7
```

[7] ✓ 37 sec Command executed in 37 sec 253 ms by arr.nagaraj on 10:49:54 PM, 7/17/22

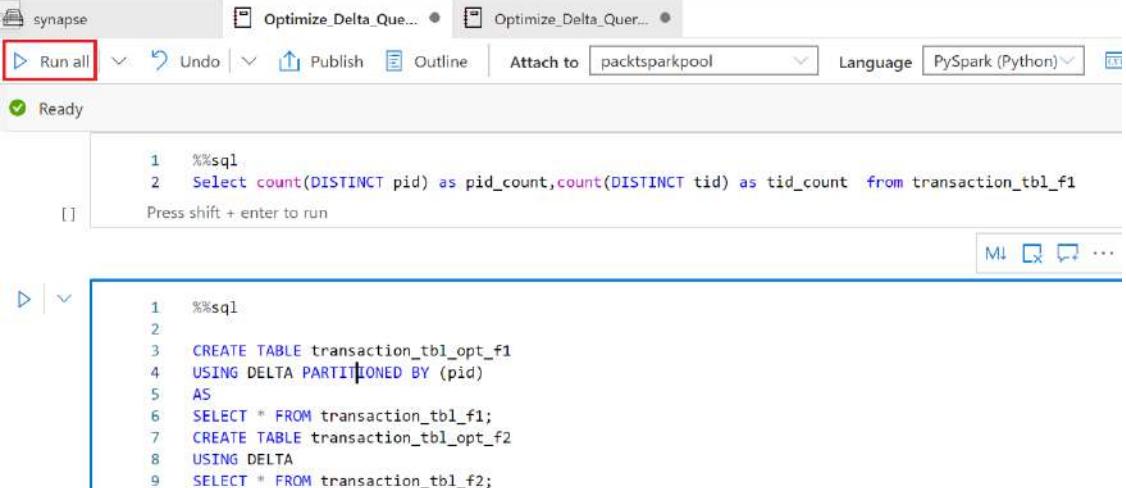
› Job execution Succeeded **Spark** 2 executors 8 cores

View Table Chart Export results ▾

pid	sum(CAST(order_count AS DOU...)
7	1840609280
3	1847656448
5	1840822272
6	1843316736
4	1853331456

Figure 12.49 – Query execution time – not optimized

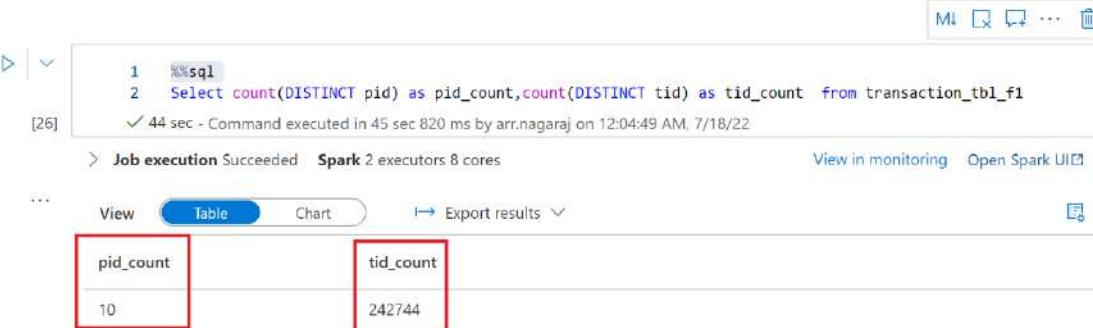
5. Download the `Optimize_Delta_Queries_Zorder_Partition.ipynb` notebook from <https://github.com/PacktPublishing/Azure-Data-Engineering-Cookbook-2nd-edition/blob/main/chapter12>. Import the notebook into Synapse Studio, as you did in steps 1 and 2 in the *How to do it...* section of the *Optimizing Delta tables in a Synapse Spark Pool lake database* recipe. Go to the `Optimize_Delta_Queries_Zorder_Partition` notebook. Select `packtsparkpool` under **Attach to** and click **Run all**:



```
synapse Optimize Delta Que... Optimize Delta Quer...
Run all Undo Publish Outline Attach to packtsparkpool Language PySpark (Python)
Ready
1 %%sql
2 Select count(DISTINCT pid) as pid_count, count(DISTINCT tid) as tid_count from transaction_tbl_f1
Press shift + enter to run
Ml Q C ...
D | v
1 %%sql
2
3 CREATE TABLE transaction_tbl_opt_f1
4 USING DELTA PARTITIONED BY (pid)
5 AS
6 SELECT * FROM transaction_tbl_f1;
7 CREATE TABLE transaction_tbl_opt_f2
8 USING DELTA
9 SELECT * FROM transaction_tbl_f2;
```

Figure 12.50 – Running the optimized notebook

6. Observe the result in the first cell, as shown in the following screenshot:



pid_count	tid_count
10	242744

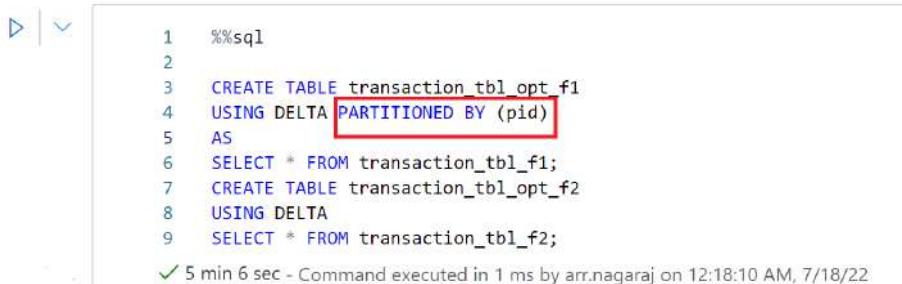
Figure 12.51 – Distinct row count

We know that the `pid` column will contain 10 distinct values and that the `tid` column will contain approximately 250,000 distinct values. A column participating in the `WHERE` clause or `JOIN` condition is a good candidate for partitioning. However, the column that's used for partitioning should not have too many distinct values as it will create too many files and subfolders. Also, there should be at least 1 million rows per partition. We know that `transaction_tbl_f1` contains 25 million rows and that there are only 10 distinct values for the `pid` column in the `transaction_tbl_f1` table. So, for each value of `pid`, we should have around 2.5 million rows on average. Observe the following query that we want to tune:

```
Select t1.pid,sum(t2.order_count)
FROM transaction_tbl_f1 t1
inner join transaction_tbl_f2 t2 on t1.tid = t2.tid
WHERE t1.pid between 3 and 7
Group by t1.pid
```

Here, we can see that the `pid` column is participating in the `WHERE` clause, so the `pid` column would make a good fit for the partition column as it has fewer distinct values and is participating in a `WHERE` clause.

7. Observe the second cell in the notebook. This call creates a table called `transaction_tbl_opt_f1` that's been partitioned by the `pid` column using `transaction_tbl_f1`. The `transaction_tbl_opt_f2` table has been created without partitioning from `transaction_tbl_f2` as it is a smaller table that only contains 500,000 rows; partitioning it would result in small subfolders, which is not recommended:



```
1 %%sql
2
3 CREATE TABLE transaction_tbl_opt_f1
4 USING DELTA PARTITIONED BY (pid)
5 AS
6 SELECT * FROM transaction_tbl_f1;
7 CREATE TABLE transaction_tbl_opt_f2
8 USING DELTA
9 SELECT * FROM transaction_tbl_f2;
```

✓ 5 min 6 sec - Command executed in 1 ms by arr.nagaraj on 12:18:10 AM, 7/18/22

Figure 12.52 – Creating partitioned tables

8. The **transaction_tbl_opt_f1** and **transaction_tbl_opt_f2** tables are joined using the **tid** column. Columns participating in the join condition can be optimized by **Z-ordering**. Z-ordering helps locate columns with the same value in different Parquet files. In our query, as we need to join **transaction_tbl_opt_f1** and **transaction_tbl_opt_f2** with the **tid** column, Z-ordering both the tables using the **tid** column will help the Delta engine identify the matching rows easily. Z-ordering is only effective when you have significant distinct values in the column; therefore, the **tid** column will be a good fit for Z-ordering as it contains approximately 250,000 rows. Cell 3 in the notebook performs Z-ordering, as shown in the following screenshot:

+ Code + Markdown

```
1 %%sql
2 Optimize transaction_tbl_opt_f1 zorder by (tid);
3 Optimize transaction_tbl_opt_f2 zorder by (tid);
```

✓ 7 min 45 sec - Command executed in < 1 ms by arr.nagaraj on 12:25:56 AM, 7/18/22

View Table Chart Export results ▾

name	value
numFilesBefore	2170
numBytesBefore	6418144533
numFilesAfter	10

Figure 12.53 – Z-ordering

9. Cell 4 of the notebook reruns the queries but now with the **transaction_tbl_opt_f1** and **transaction_tbl_opt_f2** tables as they have been partitioned and Z-ordered. The query was completed in 26 seconds compared to the earlier run, which took 37 seconds:

+ Code + Markdown

```
1 %%sql
2
3 Select t1.pid,sum(t2.order_count)
4 FROM transaction_tbl_opt_f1 t1
5 inner join transaction_tbl_opt_f2 t2 on t1.tid = t2.tid
6 WHERE t1.pid between 3 and 7
7 Group by t1.pid
```

✓ 26 sec - Command executed in 27 sec 61 ms by arr.nagaraj on 12:26:23 AM, 7/18/22

Job execution Succeeded Spark 2 executors 8 cores

Figure 12.54 – The optimized result

How it works...

Partitioning the **transaction_tbl_opt_f1** table using **pid** organizes the table with a subfolder for each value of **pid**. Instead of all Parquet files in one folder named **transaction_tbl_opt_f1**, the Parquet files were placed in 10 subfolders inside the **transaction_tbl_opt_f1** folder in the data lake, which makes it easier for the Spark engine to read only the folders containing Parquet files that satisfy the **t1.pid** filter condition between 3 and 7. In Synapse Studio, click on the storage icon (the cylinder-like symbol on left). Then, click on the **Linked** tab, expand **packtadesynapse**, click on **synapse (Primary)**, and navigate to the **synapse | workspaces | packtadesynapse | warehouse | transaction_tbl_opt_f1** folder. The folder should look as follows:

The screenshot shows the Microsoft Synapse Studio interface. On the left, there's a sidebar with a storage icon (cylinder) highlighted with a red box. Below it, the 'Data' section has the 'Linked' tab selected, also highlighted with a red box. The main area shows a file tree and a detailed list of subfolders under 'transaction_tbl_opt_f1'. The file tree on the left shows 'Azure Data Lake Storage Gen2' with a red box around it, containing 'packtadesynapse (Primary - packta...' and 'synapse (Primary)'. The 'transaction_tbl_opt_f1' folder is expanded, showing subfolders for '_delta_log' and 'pid=1' through 'pid=9', all highlighted with red boxes. The list view on the right shows these subfolders with columns for Name, Last Modified, and Content Type.

Name	Last Modified	Content Type
_delta_log	18/07/2022, 00:55:35	Folder
pid=1	18/07/2022, 00:55:41	Folder
pid=10	18/07/2022, 00:55:42	Folder
pid=2	18/07/2022, 00:55:43	Folder
pid=3	18/07/2022, 00:55:44	Folder
pid=4	18/07/2022, 00:55:44	Folder
pid=5	18/07/2022, 00:55:45	Folder
pid=6	18/07/2022, 00:55:46	Folder
pid=7	18/07/2022, 00:55:47	Folder
pid=8	18/07/2022, 00:55:47	Folder
pid=9	18/07/2022, 00:55:48	Folder

Figure 12.55 – subfolders – partitioning

Z-ordering helps identify the matching rows that have the same value for the **tid** column in both tables. Z-ordering needs to be scheduled regularly for it to be effective.

There are other techniques to optimize Delta lake query performance, such as indexing, caching, and many more. You may refer to <https://docs.microsoft.com/en-us/azure/synapse-analytics/spark/apache-spark-performance> to explore other techniques.

13

Monitoring and Maintaining Azure Data Engineering Pipelines

12 chapters and 500+ pages into this book should have given you a fair understanding of building Data Factory/Synapse integration pipelines to ingest, process, and store your data. The next key aspect of Azure data engineering is to monitor and maintain the pipelines that you've developed.

In this chapter, we will cover the following recipes:

- Monitoring Synapse integration pipelines using Log Analytics and workbooks
- Tracing SQL queries for Synapse dedicated SQL pools to Synapse integration pipelines
- Provisioning a Microsoft Purview account and creating a data catalog
- Integrating a Synapse workspace with Microsoft Purview and tracking data lineage
- Applying Azure tags using PowerShell to multiple Azure resources

After completing these recipes, you will be familiar with integrating Synapse workspaces with Log Analytics, monitoring Synapse pipeline performance and failures, using labels to trace queries fired by Synapse pipelines on dedicated SQL pools, tracking data lineage using Microsoft Purview, Purview integration with Synapse, and bulk tagging resources using PowerShell.

Technical requirements

For this chapter, you will need the following:

- A Microsoft Azure subscription
- PowerShell 7 and above
- Follow the *Install the Azure Az PowerShell module* instructions at <https://docs.microsoft.com/en-us/powershell/azure/install-az-ps?view=azps-8.0.0>

Monitoring Synapse integration pipelines using Log Analytics and workbooks

One of the key tasks after developing complex Synapse integration pipelines is to monitor them for aspects such as long-running activities, large data movements, and common errors. In this recipe, we will integrate a Synapse workspace with Azure Log Analytics and deploy a ready-made workbook to monitor the pipelines. In this recipe, at a high level, we will be performing the following tasks:

- Import a pipeline from a template. The pipeline's execution data will be monitored using an Azure Log Analytics workspace and workbook.
- Integrate a Synapse workspace with Azure Log Analytics.
- Deploy a workbook for monitoring the Synapse integration pipeline.

Getting ready

To get started, log into <https://portal.azure.com> using your Azure credentials and perform the following steps:

1. Create a Synapse Analytics workspace, as explained in the *Provisioning an Azure Synapse Analytics workspace* recipe of *Chapter 8, Processing Data Using Azure Synapse Analytics*.
2. Download the `transaction-tbl.csv` file from <https://github.com/PacktPublishing/Azure-Data-Engineering-Cookbook-2nd-edition/tree/main/chapter10>. In the Synapse Analytics workspace, create two folders named CSV and Parquet in the data lake account attached to it. Upload the `transaction-tbl.csv` file to the CSV folder. For detailed screenshots of a similar task, follow steps 1 to 4 in the *How to do it...* section of *Analyzing Data using a Serverless SQL pool* recipe in *Chapter 8, Processing Data Using Azure Synapse Analytics*.

3. Create a Synapse dedicated SQL pool named `packtadesqlpool`, as described in *steps 1 to 3* in the *How to do it...* section of the *Loading data into a dedicated SQL pool using PolyBase and T-SQL* recipe of *Chapter 10, Building the Serving Layer in Azure Synapse SQL Pool*.
4. Create a Log Analytics workspace (if you haven't done so already), as explained in *steps 1 and 2* in the *How to do it...* section of the *Configuring a Log Analytics workspace for a Synapse dedicated SQL pool* recipe of *Chapter 11, Monitoring Synapse SQL and Spark Pools*.

How to do it...

Follow these steps to monitor Synapse pipelines using a Log Analytics workbook:

1. Let's integrate the Log Analytics workspace with Synapse. Log into `portal.azure.com`, go to **All resources**, and search for `packtadesynapse`, the Synapse Analytics workspace you created previously. Find **Diagnostic settings** under the **Monitoring** section on the left and click on **Add diagnostic setting**:

Home > PacktADESynapse > packtadesynapse

packtadesynapse | Diagnostic settings

Synapse workspace | Directory: Microsoft

Dia

Diagnose and solve problems

Monitoring

Diagnostic settings

Support + troubleshooting

New Support Request

Refresh Feedback

Diagnostic settings

Name	Storage account
No diagnostic settings defined	

+ Add diagnostic setting

Click 'Add Diagnostic setting' above to configure the collection o

Figure 13.1 – Configuring the Log Analytics workspace

2. Set **Diagnostic setting** to **IntegrationPipeline**. Select **Integration Pipeline Runs**, **Integration Activity Runs**, and **Integration Trigger Runs** under **Categories**. Select the **Send to Log Analytics workspace** option under **Destination details** and select **PacktADELogAnalytics** under **Log Analytics workspace**. Then, click **Save**:

Home > PacktADESynapse > packtadesynapse | Diagnostic settings >

Diagnostic setting ...

Save Discard Delete Feedback

A diagnostic setting specifies a list of categories of platform logs and/or metrics that you want to collect from a resource, and one or more destinations that you would stream them to. Normal usage charges for the destination will occur. [Learn more about the different log categories and contents of those logs](#)

Diagnostic setting name *

IntegrationPipeline

Logs

Categories

- Synapse RBAC Operations
- Synapse Gateway Api Requests
- Built-in Sql Pool Requests Ended
- Integration Pipeline Runs
- Integration Activity Runs
- Integration Trigger Runs

Destination details

- Send to Log Analytics workspace
- Subscription: Visual Studio Enterprise
- Log Analytics workspace: PacktADELogAnalytics (eastus)
- Archive to a storage account
- Stream to an event hub
- Send to partner solution

Figure 13.2 – Saving the Log Analytics workspace

3. The next step is to import a sample pipeline. We will be monitoring the sample pipeline's execution and diagnostics data via the Log Analytics workspace. The sample pipeline will read the `transaction-tbl.csv` file, load it into the `packtadesqlpool` database's table using the **Copy activity** task, perform transformations using the **Data flow** task, and finally save the result as Parquet files in the Parquet folder in the data lake account. Follow these steps to import a pipeline:
- Download the `SynapsePipeline.zip` file from <https://github.com/PacktPublishing/Azure-Data-Engineering-Cookbook-2nd-edition/tree/main/chapter13>.
 - Open **Synapse Studio**. Click on the integration pipeline icon (the pipe-like symbol on the left). Click on **+** and then **Import from pipeline template**:

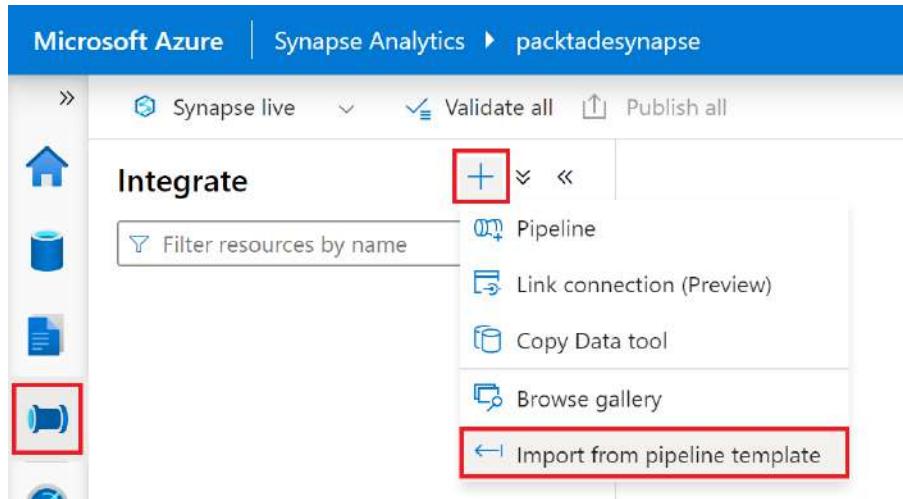


Figure 13.3 – Import from pipeline template

- Select the `SynapsePipeline.zip` file you downloaded. Then, select `packtadesynapse-WorkspaceDefaultSqlServer` for the first **Linked service** box and then `packtadesynapse-WorkspaceDefaultStorage` for the second **Linked service** box for the data lake connection, as shown in the following screenshot. Then, click **Open pipeline**:

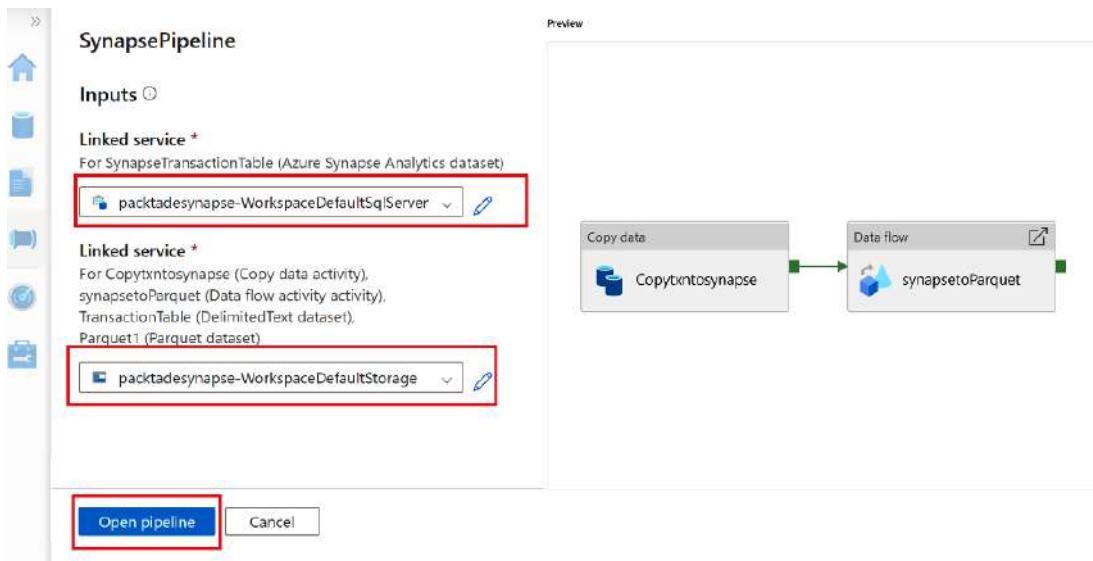


Figure 13.4 – Import pipeline – setting a linked service

- Click on the **Publish all** button:

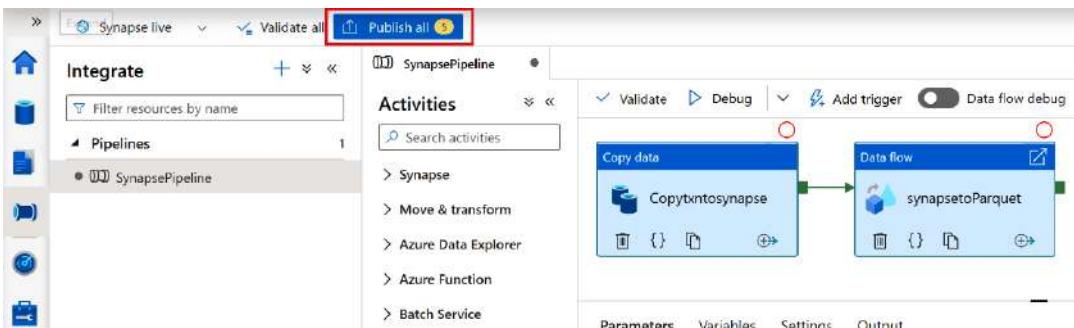


Figure 13.5 – Publishing the pipeline

- Click on **Add trigger** and select **Trigger now** to execute the pipeline. Optionally, run the pipeline two or three times if you wish to have more diagnostic/execution data for the Log Analytics workspace:

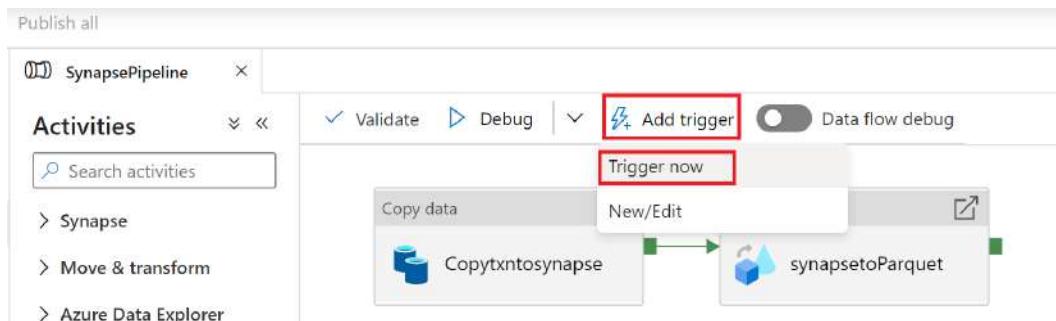


Figure 13.6 – Executing the pipeline

- Download the `SynapsePipeline.workbook` file from <https://github.com/PacktPublishing/Azure-Data-Engineering-Cookbook-2nd-edition/blob/main/chapter13>.
- Go to **All resources** and search for **PacktADELogAnalytics**, the Log Analytics workspace. Under the **General** section, click on **Workbooks**. Use the contents of `SynapsePipeline.workbook` and create a new workbook, as explained in steps 2 to 5 in the *How to do it...* section of the *Creating workbooks in a Log Analytics Workspace to visualize monitoring data* recipe of Chapter 11, *Monitoring Synapse SQL and Spark Pools*. Once done, select **Log Analytics Subscription** as your subscription, **Log Analytics workspace** as **PacktADELogAnalytics**, and **Synapse/Datafactory** as **packtadesynapse**. Then, click **Done Editing**:

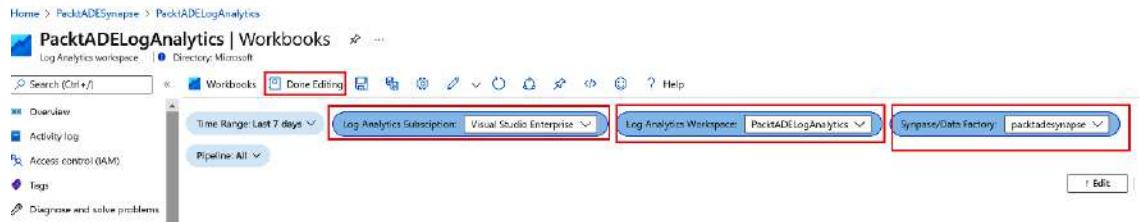


Figure 13.7 – Pipeline monitoring workbook

6. Click on the **Save** button and use **SynapsePipeline** as the workbook's name:

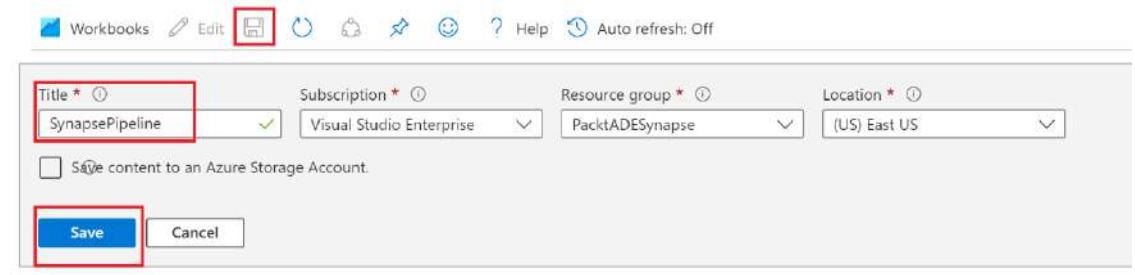


Figure 13.8 – Saving the pipeline

7. **Overview Dashboard** specifies the average duration of pipelines in the workspace and insight into the successes/failures of pipelines:



Figure 13.9 – Pipeline monitoring dashboard

8. The **Pipeline Details** tab provides useful insights into information such as error trends, count of error messages by type, and recent failure details. However, as our workspace has been just created, it doesn't have enough data to display, hence why it's empty:

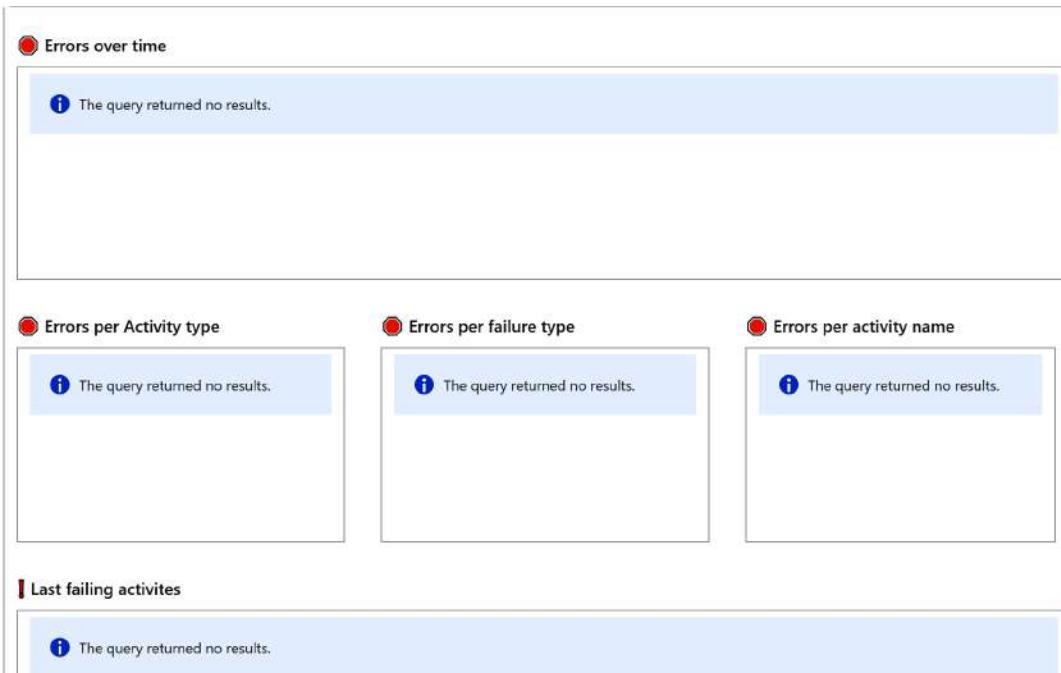


Figure 13.10 – Pipeline monitoring – error details

Refer to https://github.com/microsoft/Azure_Synapse_Toolbox/blob/master/Collateral/Screenshots/SynapsePipelineWorkbook1.png for an example dashboard where failures were recorded.

How it works...

A Log Analytics workspace, once integrated with an Azure Synapse workspace for the Integration Activity/Trigger/Pipeline runs categories, collects execution data for all integration pipelines in the Synapse workspace. The workbook presents the data collected by Log Analytics in a visually interactive way. The workbook that was used in this recipe can be found at https://github.com/microsoft/Azure_Synapse_Toolbox/tree/master/Monitor_Workbooks, the official GitHub page maintained by Microsoft for Synapse.

The best part about the solution is that even when new additional pipelines are added to the Synapse workspace, the workbook in Log Analytics will automatically reflect the execution data of new pipelines. Monitoring for error trends and pipeline duration becomes extremely critical when you have hundreds of pipelines, with each pipeline containing several tasks and complex workflows. This workbook will be very handy in identifying errors and performance issues and addressing them proactively.

Tracing SQL queries for dedicated SQL pool to Synapse integration pipelines

In the *Using Kusto queries to monitor SQL and Spark pools* recipe of *Chapter 11, Monitoring Synapse SQL and Spark Pools*, we explored using Kusto queries and a Log Analytics workspace to find expensive queries in a dedicated SQL pool. However, in data engineering projects, finding the expensive queries in a dedicated SQL pool alone wouldn't be sufficient as you need to find the details about the integration pipeline that fired the query. To do this, we need to find a way to correlate the Log Analytics data from the integration pipelines and a dedicated SQL pool.

Fortunately, **Copy activity** in an integration pipeline automatically adds a label to the SQL query it uses to copy the data. We can easily identify the pipeline and activity name from the label attached to the SQL query in the dedicated SQL pool. However, other activities, such as data flows and SQL stored procedure tasks, don't automatically append labels to queries, making it harder to trace the pipeline details from the query. A workaround for other activities, such as data flows, is to manually append labels to the queries fired inside them so that they can be traced back to the activity/pipeline.

In this recipe, we will use Kusto queries and a Log Analytics workspace to trace the pipeline details for queries from a **Copy activity** and showcase manually appending labels to queries in a data flow activity.

Getting ready

To get started, log into <https://portal.azure.com> using your Azure credentials. Then, follow these steps:

1. Complete all the steps listed in the *Getting ready* section of the *Monitoring Synapse integration pipelines using Log Analytics and workbooks* recipe.
2. Link the Log Analytics workspace to a dedicated SQL pool, as explained in *steps 3 to 5* in the *How to do it...* section of the *Configuring a Log Analytics workspace for a Synapse dedicated SQL pool* recipe of *Chapter 11, Monitoring Synapse SQL and Spark Pools*.
3. Complete *steps 1 to 3* in the *How to do it....* section of the *Monitoring Synapse integration pipelines using Log Analytics and workbooks* recipe to configure Log Analytics for integration pipelines and to execute a sample pipeline.

How to do it...

After completing the *Getting ready* section, you would have executed **SynapsePipeline**, which contains a **Copy** activity and **Data flow** activity. First, we will trace the pipeline and activity details of queries fired from the Copy activity.

Follow these steps to find the pipelines and activities details for queries fired from the Copy activity:

1. In the Azure portal, go to **All resources** and search for **PacktADELogAnalytics**, the Log Analytics workspace, and click on it. Click on **Logs**. Then, close the **Queries** popup using the **close** button at the top right:

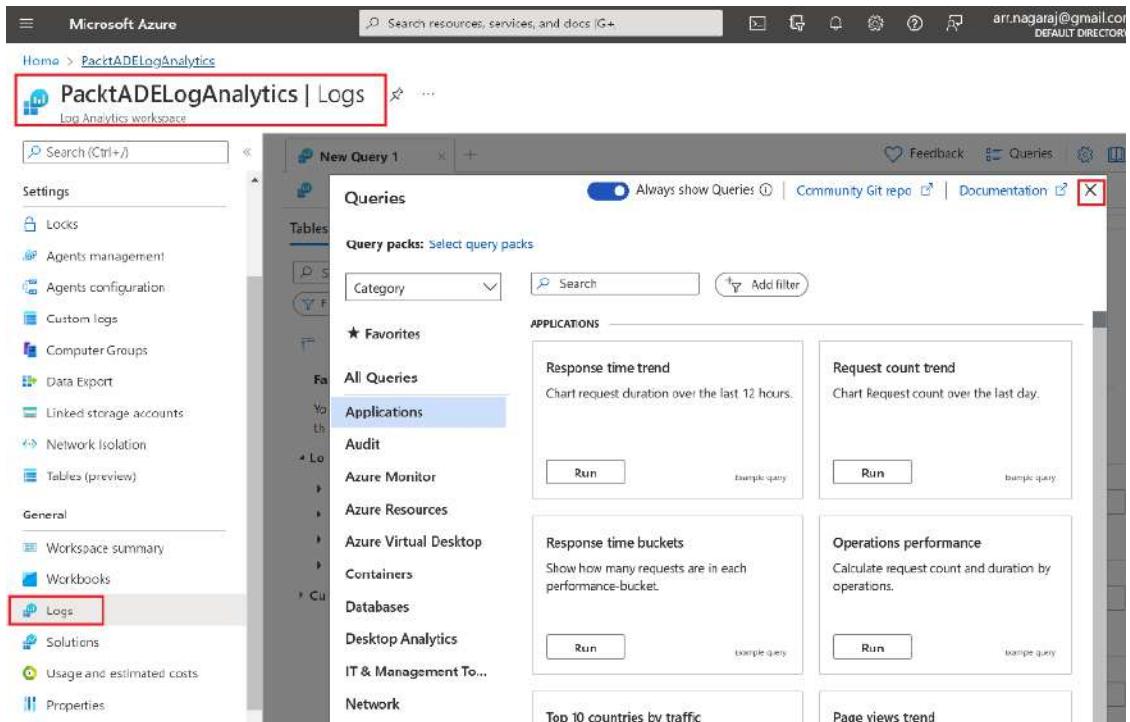


Figure 13.11 – Closing the Queries popup

2. In the query window, paste the following script:

```
let SQLQueries =
    SynapseSqlPoolExecRequests
    | where StartTime between (datetime(2000-05-20) ..
    TimeGenerated)
    | where Label != "health_checker" and Label contains "ADF"
    | where Status contains "Running"
    | where _ResourceId endswith DatabaseName
```

```
| extend duration_sec = datetime_diff("second",
TimeGenerated,StartTime)
| summarize Query_duration_sec = max(duration_
sec),StartTime = min(StartTime), Command =
any(Command),Label = any(Label),ResourceClass =
any(ResourceClass),QueryPlan = any(ExplainOutput),Status
= any(Status),Source = any(SourceSystem) by RequestId;
```

The preceding script finds the queries on a dedicated SQL pool that have labels that contain the word ADF. Copy the activity in the Synapse integration pipeline; Azure Data Factory will automatically add a label in "ADF Activity: <Activity ID>" format. The Activity ID is a unique identifier that you can use to identify whether an activity has been executed inside a Synapse integration pipeline or a Data Factory pipeline.

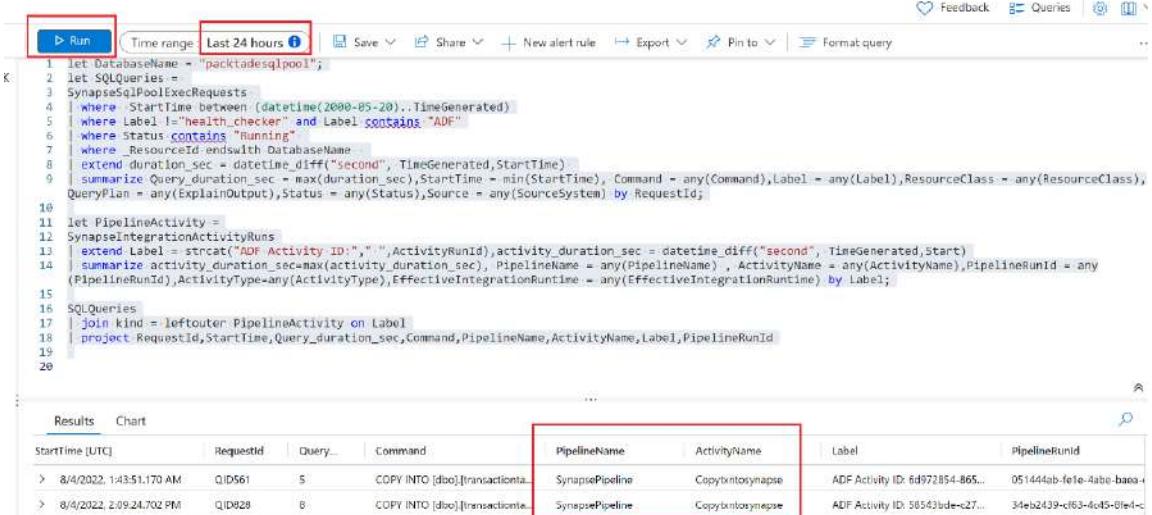
Append the following script to the same query window:

```
let PipelineActivity =
SynapseIntegrationActivityRuns
| extend Label = strcat("ADF Activity ID:", "
",ActivityRunId),activity_duration_sec = datetime_diff
("second", TimeGenerated,Start)
| summarize activity_duration_sec=max(activity_duration_
sec), PipelineName = any(PipelineName) , ActivityName =
any(ActivityName),PipelineRunId = any(PipelineRunId),
ActivityType=any(ActivityType),
EffectiveIntegrationRuntime = any(EffectiveIntegration
Runtime) by Label;
```

The preceding script gets the pipeline's name, run ID, activity name, and the activity IDs of the pipelines that were executed. Append the following script to the same query window:

```
SQLQueries
| join kind = leftouter PipelineActivity on Label
| project RequestId,StartTime,Query_duration_
sec,Command,PipelineName,ActivityName,Label,PipelineRunId
```

The preceding script combines the SQL query details and pipeline/activity details using the activity ID, which is stored in the **Label** column. Select the time range to **Last 24 hours**, select all the queries you've pasted, and hit the **Run** button:



```

let DatabaseName = "packtadesqlpool";
let SQLQueries =
SynapseSqlPoolExecRequests
| where StartTime between (datetime(2000-05-20)..TimeGenerated)
| where Label like "health_checker" and Label contains "ADF"
| where Status contains "Running"
| where _ResourceId endsWith DatabaseName
| extend duration_sec = datetime_diff("second", TimeGenerated,StartTime)
| summarize Query_duration_sec = max(duration_sec),StartTime = min(StartTime), Command = any(Command),Label = any(Label),ResourceClass = any(ResourceClass),QueryPlan = any(ExplainOutput),Status = any(Status),Source = any(SourceSystem) by RequestId;
11 let PipelineActivity =
SynapseIntegrationActivityRuns
| extend Label = strcat("ADF_Activity_ID:", "",ActivityRunId),activity_duration_sec = datetime_diff("second", TimeGenerated,Start)
| summarize activity_duration_sec=max(activity_duration_sec),PipelineName = any(PipelineName), ActivityName = any(ActivityName),PipelineRunId = any(PipelineRunId),ActivityType=any(ActivityType),EffectiveIntegrationRuntime = any(EffectiveIntegrationRuntime) by Label;
16 SQLQueries
| join kind = leftouter PipelineActivity on Label
| project RequestId,StartTime,Query_duration_sec,Command,PipelineName,ActivityName,Label,PipelineRunId

```

StartTime [UTC]	RequestId	Query...	Command	PipelineName	ActivityName	Label	PipelineRunId
> 8/4/2022, 1:43:51.170 AM	QID561	5	COPY INTO [dbo].[transactionta...	SynapsePipeline	CopyIntoSynapse	ADF Activity ID: 6d972854-865...	051444ab-fcfe-4abe-ba0...
> 8/4/2022, 2:09:24.702 PM	QID828	8	COPY INTO [dbo].[transactionta...	SynapsePipeline	CopyIntoSynapse	ADF Activity ID: 56543bd8-c27...	34eb2439-cf63-4c45-0led-c...

Figure 13.12 – Activity and pipeline details

As you can see, the script provides the pipeline's name and the name of the Copy activity that fired the query on the dedicated SQL pool. Using **PipelineRunId**, we can find the exact execution of the pipeline that fired the query.

As explained initially, data flows don't automatically add labels; they need to be appended manually to queries from data flow. Next, we will learn how to manually add a label to trace the activity's details.

3. Go to **Synapse Studio**. Click on the develop icon (the notebook-like icon), click on the + button, and select **SQL script**. Then, connect to **packtadesqlpool**, copy and paste the following script, and execute it:

```

CREATE PROC [dbo] . [get_transactiontable_df] AS

    Select * from dbo.transactiontable
    Option (LABEL = 'ADF: SQLtoParquet - Dataflow' )

```

The preceding script creates a stored procedure that contains a SELECT query that reads all the rows from **dbo.transactiontable**. The SELECT query uses an OPTION clause that adds the label to the query:

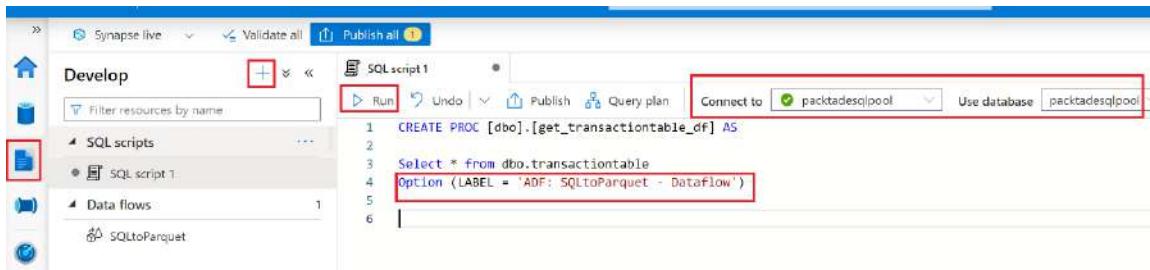


Figure 13.13 – Stored procedure creation

4. Expand **Data flows** and click on the **SQLtoParquet** data flow. Then, click on **source1**, then **Source options**. Select **Stored procedure** under **Input** and set **Schema name** to **dbo**. Finally, set **Stored procedure** to **get_transactiontable_df**, the name of the stored procedure you created in step 3. Then, click on **Publish all**:

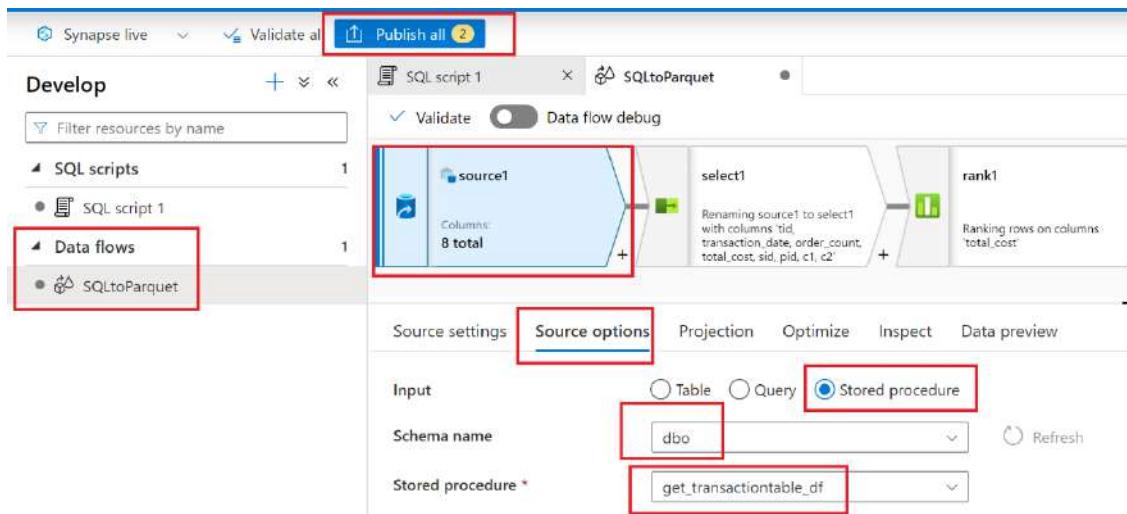


Figure 13.14 – Stored procedure input

Previously, we used **Table** as the input option, which obtained all the rows from **dbo.transactiontable**. We have set **Stored procedure** to **Input** here, which allows us to manually append the label and fetch all the rows from the table.

5. Click on **Integration Pipeline Icon**, go to **SynapsePipeline**, and execute **SynapsePipeline** by clicking on **Add trigger | Trigger now**:

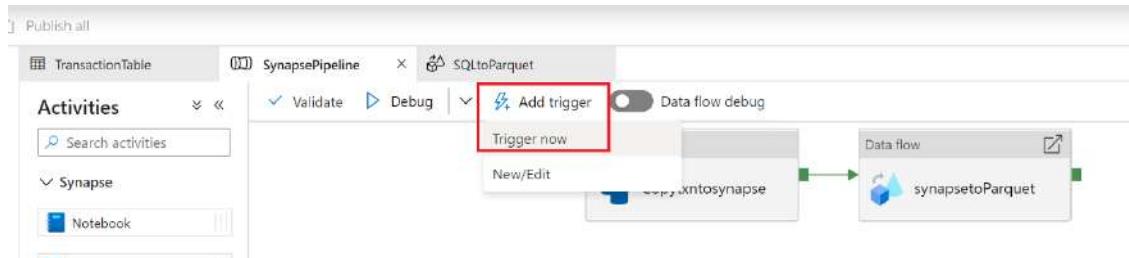


Figure 13.15 – Triggering the pipeline

6. Rerun the KQL script you used in *step 2* in the Azure Log Analytics workspace. Now, the query from the data flow is also listed along with the query from the Copy activity. The label we added in the stored procedure is listed in the result set. Using that label, we can identify that the query was fired from the data flow:

RequestID	StartTime [UTC]	Query_du...	Command	PipelineName	ActivityName	Label	PipelineRunId
> QID1688	8/5/2022, 5:31:51.672 AM	6	COPY INTO [dbo].[transactionta...]	SynapsePipeline	CopytoSynapse	ADF Activity ID: e6cbcf9-0fd3-...	bfe2d450-2073-4821-b6e8-bc5...
> QID1750	8/5/2022, 5:36:10.254 AM	6	Select * from dbo.transactionta...			ADF:SQLtoParquet - Dataflow	
> QID1743	8/5/2022, 5:36:08.029 AM	1	Select * from dbo.transactionta...			ADF:SQLtoParquet - Dataflow	
> QID1736	8/5/2022, 5:35:58.613 AM	5	Select * from dbo.transactionta...			ADF:SQLtoParquet - Dataflow	
> QID1729	8/5/2022, 5:35:54.816 AM	9	Select * from dbo.transactionta...			ADF:SQLtoParquet - Dataflow	
> QID1722	8/5/2022, 5:35:40.941 AM	5	Select * from dbo.transactionta...			ADF:SQLtoParquet - Dataflow	

Figure 13.16 – Data flow labels listed

How it works...

Integrating a Synapse integration pipeline and a dedicated SQL pool in the same Log Analytics workspace allows us to easily correlate the diagnostic data from both services. Query labels allow us to tag queries so that we can locate them during database monitoring and troubleshooting tasks. The Copy activity automatically adds labels to queries, which helps us identify the queries fired by it easily. Other activities don't automatically add labels to queries. Hence, for the data flow activity, we manually added a label using a stored procedure and tracked the query in the Log Analytics workspace. It is recommended that you label the queries while developing data flows as it will make monitoring easier, especially in a complex environment that contains hundreds of pipelines.

Provisioning a Microsoft Purview account and creating a data catalog

Microsoft Purview is a data governance service that allows you to manage all your data assets in one single place. Using Microsoft Purview, you can build data catalogs to maintain various information about data assets such as dataset details, schema details of the dataset, data source details, dataset classification, owners of the dataset, and many more.

In this recipe, we will provision a Purview account and build a basic data catalog. After that, we will register a resource group and scan the resources/assets in the resource group. By doing this, Purview will collect and store the metadata information about all the data assets in the resource group and build a basic data catalog.

Getting ready

To get started, log into <https://portal.azure.com> using your Azure credentials. Then, follow these steps:

1. Create a Synapse Analytics workspace, as explained in the *Provisioning a Azure Synapse Analytics workspace* recipe of *Chapter 8, Processing Data Using Azure Synapse Analytics*.
2. Download the `transaction-tbl.csv` file from <https://github.com/PacktPublishing/Azure-Data-Engineering-Cookbook-2nd-edition/tree/main/chapter10>. In a Synapse Analytics workspace, create two folders named CSV and Parquet in the data lake account attached to it. Upload the `transaction-tbl.csv` file to the CSV folder. For detailed screenshots for a similar task, follow steps 1 to 4 in the *How to do it...* section of the *Analyzing data using a Serverless SQL pool* recipe of *Chapter 8, Processing Data Using Azure Synapse Analytics*.

3. Create a Synapse-dedicated SQL pool and name it `packtadesqlpool`, as described in steps 1 to 3 in the *How to do it...* section of the *Loading data into a dedicated SQL pool using PolyBase and T-SQL* recipe of Chapter 10, *Building the Serving Layer in Azure Synapse SQL Pool*.
4. Complete step 3 in the *How to do it...* section of the *Monitoring Synapse integration pipelines using Log Analytics and workbooks* recipe to deploy a sample pipeline called **SynapsePipeline** and execute it.

How to do it...

Follow these steps to provision a Purview account, register services, and scan assets:

1. First, create a Purview account. Go to `portal.azure.com` and click on **Create a resource**. Search for **Microsoft Purview** and click on the **Create** button:

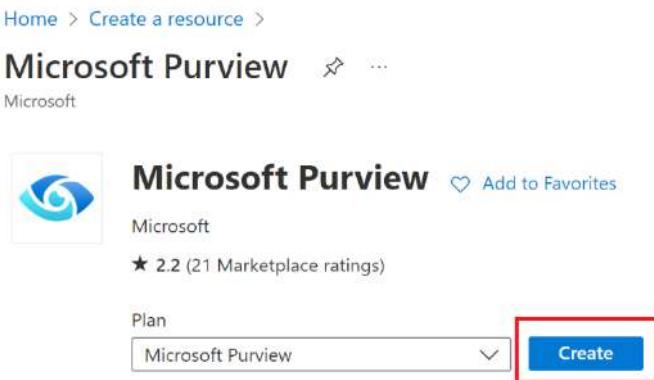


Figure 13.17 – Purview – Create a resource

2. Set **Resource group** to `PacktADESynapse` and **Microsoft Purview account name** to `packtadepurview`. Then, click **Review + Create**:

Home > Create a resource > Microsoft Purview >

Create Microsoft Purview account

Provide Microsoft Purview account info

* Basics * Networking Tags Review + Create

Create a Microsoft Purview account to develop a data governance solution in just a few clicks. A storage account and eventhub will be created in a managed resource group in your subscription for catalog ingestion scenarios. [Learn more](#)

Project details

Subscription *

Visual Studio Enterprise

Resource group *

PacktADESynapse

[Create new](#)

Instance details

Microsoft Purview account name *

packtadepurview

Location *

East US

i 1 Capacity unit (CU) = 25 ops/sec and 10 GB of metadata storage. Any new Microsoft Purview account will be provisioned with 1 CU with auto scale capabilities. [Learn more](#)

Managed resources

A resource group, a storage account, and an Eventhub will be created in the selected subscription for catalog ingestion scenarios. The Microsoft.Storage and Microsoft.EventHub resource providers will get registered. [Learn more](#)

Managed resource group name *

managed-rg-packtadepurview

Storage account name

Name will be auto-generated during account creation.

Event Hubs namespace name

[Enable](#) [Disable](#)

[Review + Create](#)

[Previous](#)

[Next: Networking >](#)

Figure 13.18 – Provisioning Purview

3. Once the Purview account has been provisioned, go to it. Click on **Open Microsoft Purview Governance Portal**:

The screenshot shows the Microsoft Purview Governance Portal interface. On the left, there's a navigation sidebar with links like Overview, Activity log, Access control (IAM), Tags, Diagnose and solve problems, Root collection permission, Settings, Managed resources, Networking, Managed identities (preview), Properties, Locks, Monitoring, Alerts, Metrics, and Diagnostic settings. The main content area displays account details: Resource group: PacktADESynapse, Status: Succeeded, Location: East US, Subscription: Visual Studio Enterprise, Subscription ID: [redacted], and Tags: [edit] with a link to Click here to add tags. Below this is a 'Get Started' section with a note: 'All roles to access Microsoft Purview Governance Portal are assigned by Microsoft Purview account collection admin in [redacted]'. It features two cards: 'Open Microsoft Purview Governance Portal' (with a red border) and 'Manage users'. The 'Open Microsoft Purview Governance Portal' card contains the text: 'Start using the unified data governance service and manage your hybrid data estate.' and a 'Open' button. The 'Manage users' card contains the text: 'Grant users access to open Microsoft Purview Governance Portal.' and a 'Go to Access control' button.

Figure 13.19 – Open Microsoft Purview Governance Portal

4. Click on the **Data map** icon (the second icon on the left) and click **Register**:

The screenshot shows the Microsoft Purview Sources page. The left sidebar has icons for Sources, Collections, Monitoring, Source management, Scan rule sets, and Pattern rules. The 'Sources' icon is highlighted with a red box. The main content area is titled 'Sources' and contains a 'Register' button (also highlighted with a red box), a 'Refresh' button, and tabs for 'Map view' and 'Table view'. Below these are buttons for 'Filter by keyword' and 'Showing 1 collection, 0 sources'.

Figure 13.20 – Registering resources in Purview

5. Select **Azure MULTIPLE** and click **Continue**:

Register sources

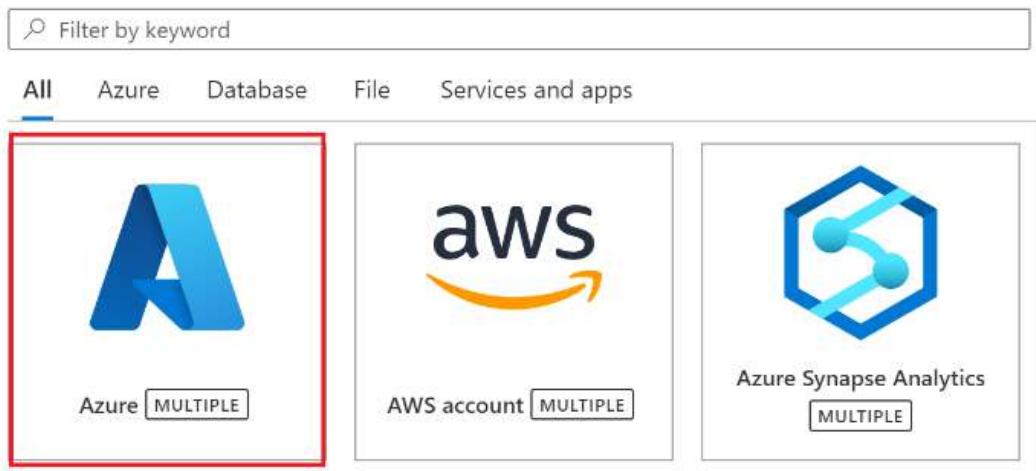


Figure 13.21 – Azure MULTIPLE

6. Provide PacktADESynapse as the name (this is the same name as the resource group to be scanned as it's easier to identify). Set both your subscription name and resource group name to PacktADESynapse. Then, click **Register**:

Register sources (Azure)

Name *
PacktADESynapse

Select from Azure subscription list or resource group list to register. Eg. You can just select a subscription and that subscription node will be registered to the graph.

Management group
Select a management group

Subscription
Visual Studio Enterprise

Resource group
PacktADESynapse

Select a collection * ⓘ
(Root) packtadepurview

Data use management
Allow data access policies to be assigned to this source. [Learn more](#) Disabled

Register **Back** **Cancel**

Figure 13.22 – Registering a resource group in Purview

7. Now that we have registered the resource group, we must scan the resources in the resource group using Microsoft Purview so that Purview can collect data about the data assets in the resource group. To do this, we need to add a **Purview managed identity** account to the following roles:
 - The **reader** role to the **packtadesynapse** workspace
 - The **Storage blob reader** role to the Azure Data Lake account
 - The **db_datareader** role to the dedicated SQL pool database

Use the following PowerShell script to add the roles for the Synapse workspace and Azure Data Lake to a Purview managed identity account. Assign the appropriate values for the \$Resourcegroup, \$Synapse_ws_name, \$storage_name, and \$purview variables:

```
$Resourcegroup = "PacktADESynapse"
$Synapse_ws_name = "packtadesynapse"
$storage_name = "packtadesynapse"
$purview = "packtadepurview"

$role = Get-AzADServicePrincipal -DisplayName $purview
$storage = Get-AzStorageAccount -ResourceGroupName
$Resourcegroup -Name $storage_name
New-AzRoleAssignment -ObjectId $role.id
-RoleDefinitionName "Storage Blob Data Reader" -Scope
$storage.id

$synapse = Get-AzSynapseWorkspace -ResourceGroupName
$Resourcegroup -Name $Synapse_ws_name
New-AzRoleAssignment -ObjectId $role.id
-RoleDefinitionName "Reader" -Scope $synapse.id
```

Upon executing the script, you will get the following output:

```
Objectid      : bde34fc0-8261-450f-83ed-e494979843fa
ObjectType    : ServicePrincipal
CanDelegate   : False
Description   :
ConditionVersion  :
Condition     :

PS C:\Users\navenvkat> $synapse = Get-AzSynapseWorkspace -ResourceGroupName $Resourcegroup -Name $Synapse_ws_name
PS C:\Users\navenvkat> New-AzRoleAssignment -ObjectId $role.id -RoleDefinitionName "Reader" -Scope $synapse.id
WARNING: We have migrated the API calls for this cmdlet from Azure Active Directory Graph to Microsoft Graph.
Visit https://go.microsoft.com/fwlink/?linkid=2181475 for any permission issues.

RoleAssignmentName : f8725c78-8cfa-4637-8b9e-26b969b4edf1
RoleAssignmentId  : /subscriptions/[REDACTED]/resourceGroups/PacktADESynapse/providers/Microsoft.Synapse/worksp
                    aces/packtadesynapse/providers/Microsoft.Authorization/roleAssignments/f8725c78-8cfa-4637-8b9e-26b969b4edf1
Scope            : /subscriptions/[REDACTED]/resourceGroups/PacktADESynapse/providers/Microsoft.Synapse/worksp
                    aces/packtadesynapse
DisplayName      : packtadepurview
SignInName      :
RoleDefinitionName : Reader
RoleDefinitionId : /subscriptions/[REDACTED]/providers/Microsoft.Authorization/roleDefinitions/acdd72a7-3385-4
                    8ef-bd42-f606fdaa2e7
ObjectId        : bde34fc0-8261-450f-83ed-e494979843fa
ObjectType      : ServicePrincipal
CanDelegate     : False
Description     :
```

Figure 13.23 – Granting permissions to PowerShell

8. Go to **Synapse Studio**. Click on the **develop** icon (the notebook-like icon). Then, click the + button and select **SQL script**. Connect to **packtadesqlpool**. Copy and paste the following script:

```
CREATE USER [packtadepurview] FROM EXTERNAL PROVIDER  
GO  
EXEC sp_addrolemember 'db_datareader', [packtadepurview]  
GO
```

Hit the **Run** button. This script will add a Purview managed identity to the **db_datareader** role on the dedicated SQL pool database:

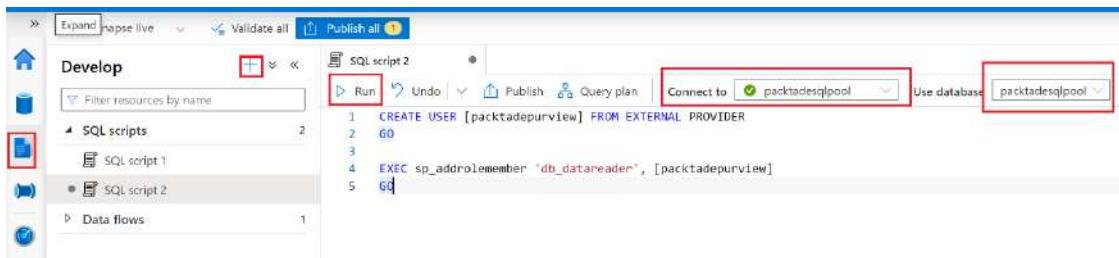


Figure 13.24 – Granting the db_datareader permission

9. Go to the **packtadepurview** Purview account and open **Microsoft Purview Governance Portal**. Click on the **Data Map** icon (the second icon on the left). Then, click the **New scan** button under **PacktADESynapse**:

The screenshot shows the Microsoft Purview interface for managing data sources. The left sidebar lists various management options: Sources (selected and highlighted with a red box), Collections, Monitoring, Source management, Scan rule sets, Pattern rules, Integration runtimes, Annotation management, Classifications, Classification rules, and Managed attributes. The main area is titled 'Sources' and displays a single collection named 'packtadepurview'. A sub-section titled 'The root collection.' is shown with a 'View details' button. Below it is an 'Azure Resource Group' named 'PacktADESynapse', which is also highlighted with a red box. This resource group has its own set of actions: 'Edit' (highlighted with a red box), 'Delete', and 'View details'. A 'New scan' button is located at the bottom of this section.

Figure 13.25 – Scanning resources

10. Select **Microsoft Purview MSI (system)** under **Credential** and click **Continue**:

Scan "PacktADESynapse"

Name *

Scan-IT7

💡 Before you set up your scan, you must give the managed identity of the Microsoft Purview account permissions to enumerate your Azure resource group. [See more](#)

Type *

All

Azure Data Lake Storage Gen2 *

All

Azure Synapse Analytics *

All

Credential *

Microsoft Purview MSI (system)

Use this credential for all types

Select a collection

(Root) packtadepurview

💡 All assets scanned will be included in the collection you select.

Continue

🔗 Test connection

Cancel

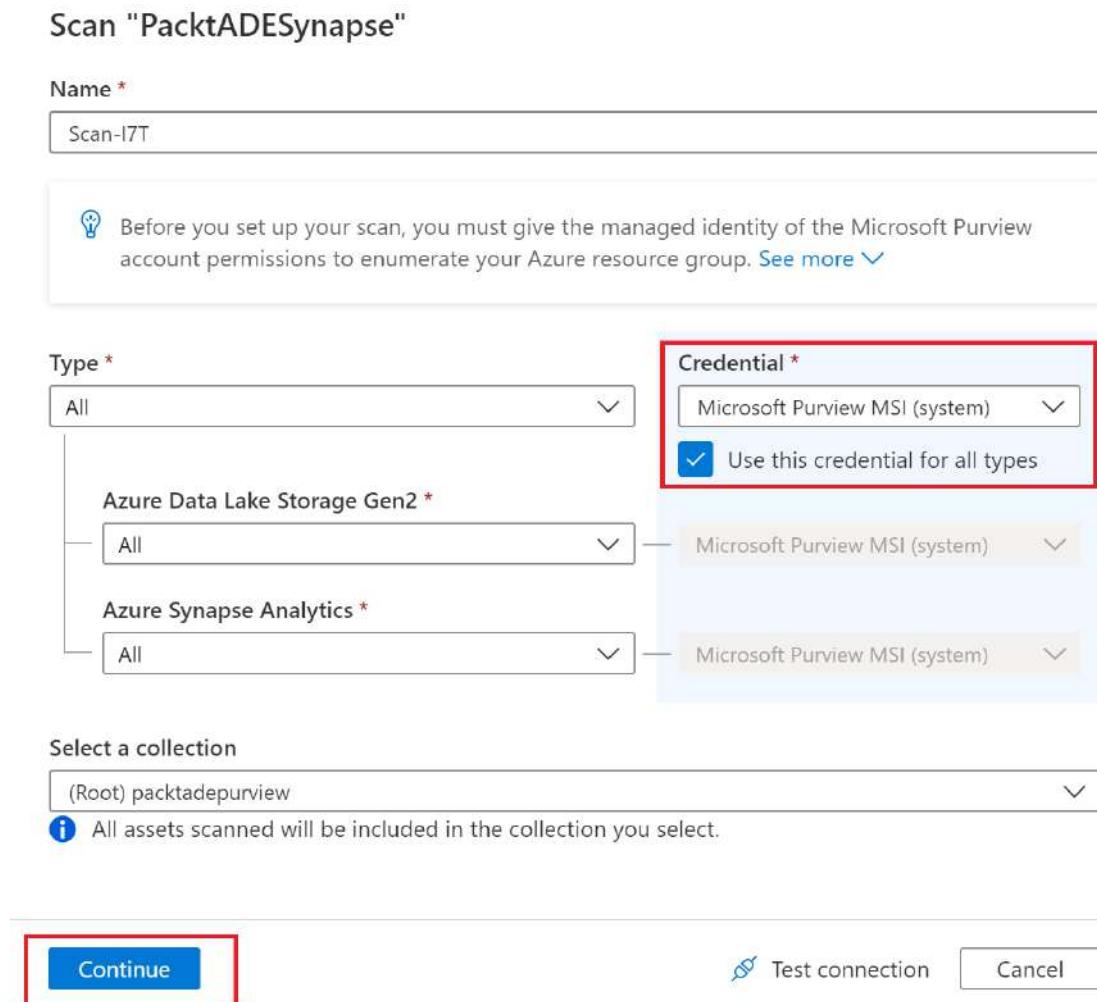


Figure 13.26 – Setting a credential

11. Ignore the **Select a scan rule set** window for now and click **Continue**:

Select a scan rule set

 New scan rule set  Refresh

Selected (2)

 AdlsGen2 SYSTEM DEFAULT

 AzureSynapseSQL SYSTEM DEFAULT

Select one scan rule for each source type. System default ones are preselected.

✓ Azure Data Lake Storage Gen2



AdlsGen2 SYSTEM DEFAULT

Microsoft default scan rule set that includes all supported file types for schema extraction and classification, and all supported system classification rules [View detail](#)

✓ Azure Synapse Analytics



AzureSynapseSQL SYSTEM DEFAULT

Microsoft default scan rule set that includes all supported system classification rules [View detail](#)

 Continue

Back

Cancel

Figure 13.27 – Skipping Select a scan rule set

12. Set the scan schedule to **Once** instead of **Recurring**. Setting the scan schedule to **Recurring** will run a resource scan at a specified time and ensure the Purview data catalog retains the updated data. For testing purposes, it is sufficient to run the scan once:

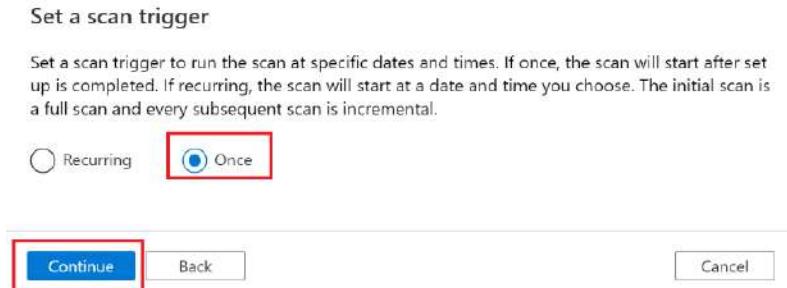


Figure 13.28 – Scanning the schedule

13. Click on **Save and Run** to run the scan:

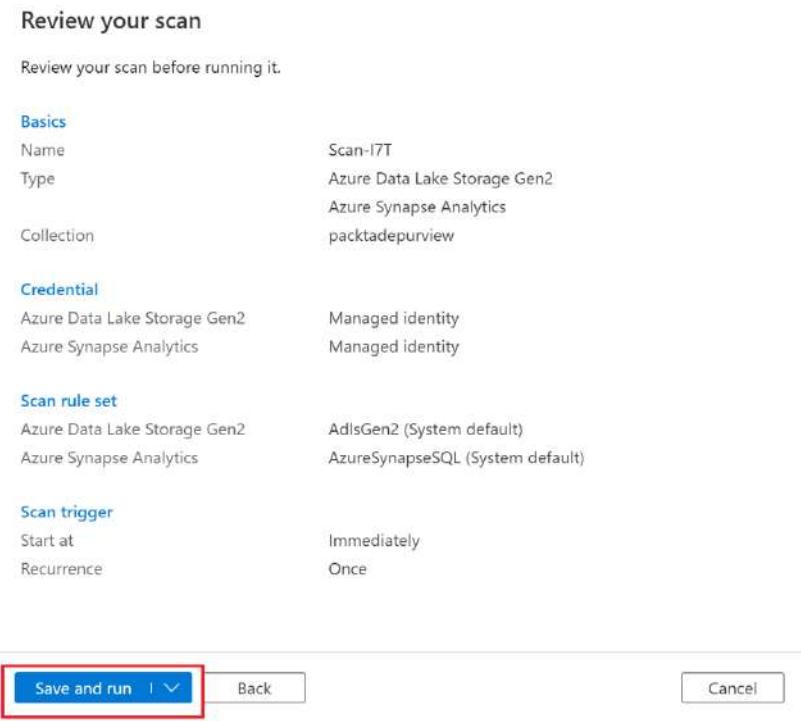


Figure 13.29 – Confirming the scan

14. Click on **View details** under **PacktADESynapse** to check the status of the scan:

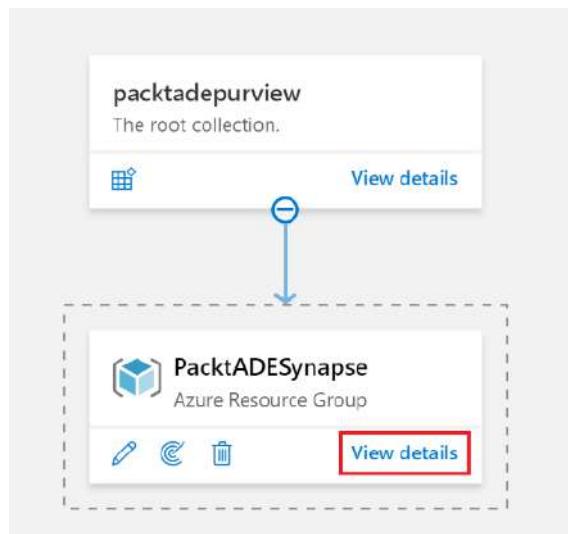


Figure 13.30 – Scan details

15. Once the run completes, you will notice that the scan has discovered assets from the resource group. Ignore the scan failure report as the error message will be related to a serverless SQL database. We can ignore it as we don't have any objects in this recipe related to a serverless SQL database. The run typically takes around 15 to 20 minutes:

A screenshot of the Microsoft Purview Data Catalog "Scan results" page. The page header shows "Data map > Sources > PacktADESynapse". Below the header, there are buttons for "New scan", "Edit source", "Delete source", and "Refresh". The "Overview" tab is selected. The main area displays the "Source ID: PacktADESynapse". It shows a summary with "Scans: 1", "Discovered assets: 3", and "Classified assets: 1", all of which are highlighted with a red box. Below this, there is a section titled "Recent scans" with a table. The table has columns: "Scan name", "Last run status", "Scan rule set", and "Last scan time". One row in the table is highlighted with a red box, showing "Scan-17T" with a failed status, "AdlsGen2, AzureSynapseSQL" as the rule set, and a timestamp of "08/05/2022, 11:11 ...".

Figure 13.31 – Scan results

16. Click on **Collections** at the top, then **packtadepurview**, and then **Assets**:

The screenshot shows the Azure Collections interface. On the left, a sidebar lists various management categories: Sources, Collections (which is selected and highlighted with a red box), Monitoring, Source management, Scan rule sets, Pattern rules, Integration runtimes, Annotation management, Classifications, Classification rules, and Managed attributes. The main pane displays the 'Collections' overview for the 'packtadepurview' collection. It includes a summary card with '4 Assets' and '1 Source'. Below this, there's a 'Description' section stating 'The root collection.' and a 'Created on' section showing 'August 5, 2022, 4:28 PM by Nagaraj Seelapudur Venkatesan'. A large 'Add a collection' button with a plus icon is prominently displayed.

Figure 13.32 – Collection details

17. The assets it has discovered will be listed here. Click on **transactiontable**:

Assets in collection		
	Name ↑	Source type
<input type="checkbox"/>	dbo	Azure Synapse Analytics
<input type="checkbox"/>	packtadesqlpool	Azure Synapse Analytics
<input type="checkbox"/>	packtadesynapse.azure.synapse.net:443	Azure Synapse Analytics
<input type="checkbox"/>	transactiontable	Azure Synapse Analytics

Figure 13.33 – Assets list

18. Click on the **Schema** tab to see the columns in the table and their data types:

The screenshot shows the Microsoft Purview Data Catalog interface. At the top, there's a breadcrumb navigation: Data catalog > Collections >. Below it, a table asset named "transactiontable" is displayed, identified as an "Azure Synapse Dedicated SQL Table". The table has 8 items. The "Schema" tab is selected and highlighted with a red box. Other tabs include Overview, Properties, Lineage, Contacts, and Related. A search bar at the top says "Filter by name". The table structure is as follows:

Column name	Classifications	Sensitivity label	Glossary terms	Data type
c1				nvarchar
c2				nvarchar
order_count				bigint
pid				bigint
sid				bigint
tid				bigint
total_cost				bigint
transaction_date				date

Figure 13.34 – Assets details

19. Click on the **Related** tab. The objects that are related to **transactiontable** are displayed here:

The screenshot shows the Microsoft Purview Data Catalog interface for the "transactiontable" asset. The "Related" tab is selected and highlighted with a red box. The related objects listed are:

- packtadesynapse.azureSynapse.net:443 (Azure Synapse Workspace)
- packtadesqlpool (Azure Synapse Dedicated SQL Database)
- dbo (Azure Synapse Dedicated SQL Schema)

A red box also highlights the list of related objects on the left side of the screen.

Figure 13.35 – Asset-related details

How it works...

The Purview account's managed identity was granted permissions on the assets in the resource group and Purview's scan was able to record all the metadata information about the assets in the resource group. Microsoft Purview creates a data catalog where you can easily locate a table or a column and find information about it, as well as list objects related to it.

You can learn more about Purview by going to the links and playing with the options on the Purview governance portal. Check out <https://docs.microsoft.com/en-us/azure/purview/use-azure-purview-studio> to learn more about the Purview governance portal.

Integrating a Synapse workspace with Microsoft Purview and tracking data lineage

One of the key features of Microsoft Purview is tracking data lineage. Microsoft Purview offers end-to-end tracking of data lineage, using which you can track the life cycle of a particular data asset – for example, if data is loaded from a CSV file into a dedicated SQL pool table and then further processed and stored in a Parquet file. Purview will be able to represent the various phases (CSV file -> SQL table -> Parquet file) of data processing graphically and automatically. Tracking data lineage in data engineering projects is critical as you need to have a bird's-eye view of the various processing phases of a data asset and its properties.

As you know, a Synapse workspace holds all the data engineering pipelines, which do all the data movement and processing. By integrating a Synapse workspace with Microsoft Purview, we can track data lineage with minimal effort.

In this recipe, we will integrate a Synapse workspace with a Microsoft Purview account and track data lineage.

Getting ready

Complete all the steps listed in the *Getting ready* section of the *Provisioning a Microsoft Purview account and creating a data catalog* recipe.

You must also complete *steps 1 to 13* in the *How to do it...* section of the *Provisioning a Microsoft Purview account and creating a data catalog* recipe.

How to do it...

Follow these steps to integrate a Synapse workspace with your Microsoft Purview account and track data lineage:

1. Log into `portal.azure.com`, go to **All resources**, and search for `packtadesynapse`. Open **Synapse Studio**. Click on the **Manage** icon (the briefcase-like icon). Click on **Microsoft Purview**, then **Connect to a Purview account**:

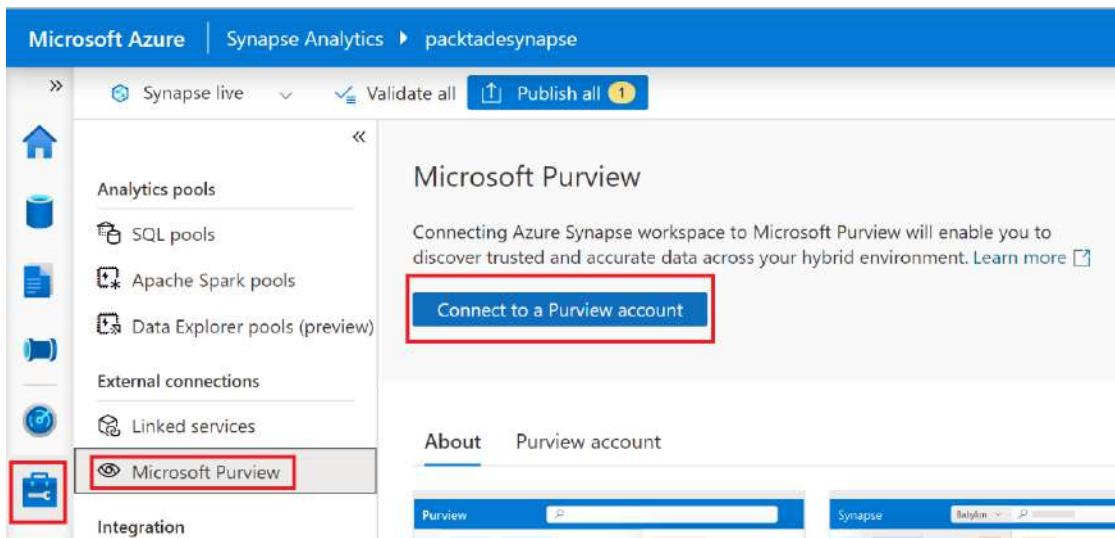


Figure 13.36 – Connect to a Purview account

2. Set **Purview account name** to `packtadepurview` and click **Apply**:

Connect to a Purview account

Connecting the Azure Synapse Analytics to Microsoft Purview will help you discover, understand, explore and share your organization's data. [Learn more](#)

Account selection method *

From Azure subscription Enter manually

Purview account name *

packtadepurview

Apply

Cancel

Figure 13.37 – Integrating Synapse with Purview

3. Click on the **Purview account** tab and ensure the connection status is **Connected**, as shown here:

The screenshot shows the Azure portal interface for managing Synapse integration. On the left, a sidebar lists various options: Data Explorer pools (preview), External connections, Linked services, Microsoft Purview, Integration (which is selected), Triggers, Integration runtimes, Security, Access control, and Credentials. The main content area has tabs for About, Purview account (which is selected and highlighted with a red box), Edit, Refresh, and Disconnect. Under the Purview account tab, it shows the account name as "packtadepurview" and a status message: "Default Purview integration capabilities: • Data Catalog - Discover enterprise data assets using Microsoft Purview powered search". Below this, it says "Additional Purview integration capabilities and status:" and shows a table with one row. The table has two columns: "Integration" and "Status". The single row contains "Data Lineage - Synapse Pipeline" in the Integration column and "Connected" with a green checkmark in the Status column, also highlighted with a red box.

Integration	Status
Data Lineage - Synapse Pipeline	Connected

Figure 13.38 – Synapse integration confirmation

4. Click on the **Integrate** icon (the pipe-like icon on the left). Expand **Pipelines** and click **SynapsePipeline**. Then, click **Add trigger**, then **Trigger now**:

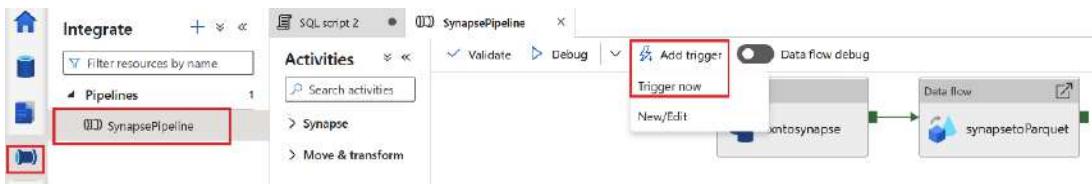


Figure 13.39 – Triggering the pipeline

5. The pipeline that was executed performs the following tasks:

- Reads a CSV file called `transaction-tbl.csv` from **packtadesynapse**, an Azure Data Lake account
- Loads the file to a table called `transactiontable` in **packtadesqlpool**, a dedicated SQL pool database that uses a Copy activity in the integration pipeline
- Reads `transactiontable`, adds a column called `rank_by_cost`, and copies the rows in Parquet format to a folder named `Parquet` in the **packtadesynapse** data lake account

Using the Purview integration, we can track the lineage from the data lake account to the Parquet folder.

Go to the **Search** area at the top of the window. Type `order_count`; Synapse will automatically prompt the `order_count` column. Click on the suggestion. `order_count` is a column that exists in the CSV file, Synapse table, and the destination Parquet file:

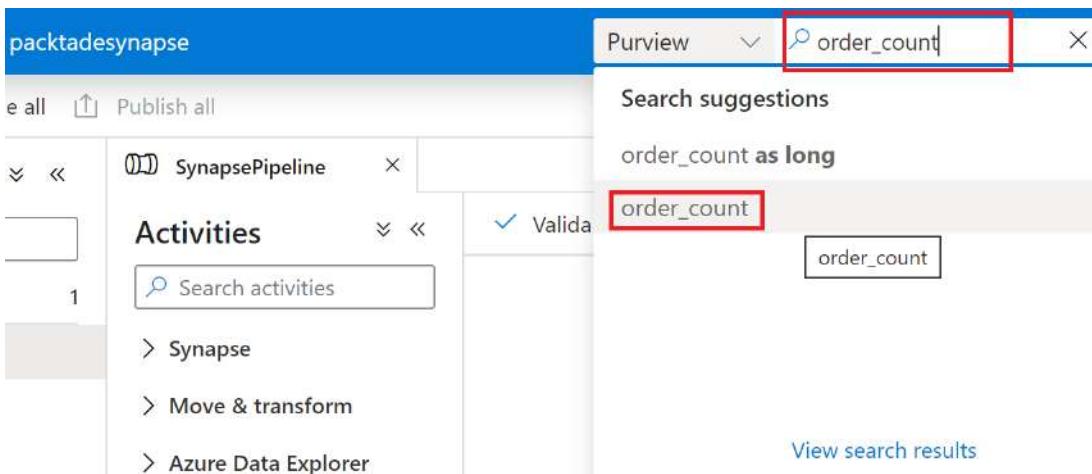


Figure 13.40 – The search column

6. All the assets that contain the **order_count** column will be listed. Click on **transactiontable**:

The screenshot shows the Azure Purview search interface with the following details:

- Search results for order_count**
- Source type : all**, **Instance : all**, **Clear all filters**
- We've improved search relevance. Try the preview**
- Showing 1-5 out of 5 results**
- Sort by: Name**
- Object Type** filter: Dashboards, Data pipelines, Files, Folders, Glossary terms, Reports, Stored procedures, Tables.
- Collection** filter: packtadepurview
- Classification** filter: ...
- Contact** filter: ...
- Label** filter: ...
- Results:**
 - SynapseImportCommand**: Azure Data Lake Storage Gen2 Resource Set, URL: https://packtadesynapse.dfs.core.windows.net/adfstagetcommandtempdata/{GUID}/SynapseImportCommand
 - synapsetoParquet**: Azure Synapse Data Flow Activity, URL: /subscriptions/bbe56a17-9802-4ca0-ba1a-f5f32c04bdee/resourceGroups/PacktADESynapse/providers/Microsoft.Synapse/works...
 - transaction tbl**: Azure Data Lake Storage Gen2 Resource Set, URL: https://packtadesynapse.dfs.core.windows.net/adfstagetcommandtempdata/{GUID}/SynapseImportCommand/transaction-tbl.txt
 - transaction-tbl.csv**: Azure Data Lake Storage Gen2 | File, URL: https://packtadesynapse.dfs.core.windows.net/synapse/CSV/transaction-tbl.csv
 - transactiontable**: Azure Synapse Dedicated SQL Table, URL: mssql://packtadesynapse.sql.azuresynapse.net/packtadesq|pool/dbo/transactiontable

Figure 13.41 – Selecting an asset

7. Click on the **Lineage** tab. The **Lineage** tab shows the data asset's complete life span. It visually shows how the data moved from the data lake account to **transactiontable** in the Synapse-dedicated pool database via the Copy activity and then to the Parquet folder via a data flow. Hover your mouse over each block to learn more about the processing phase:

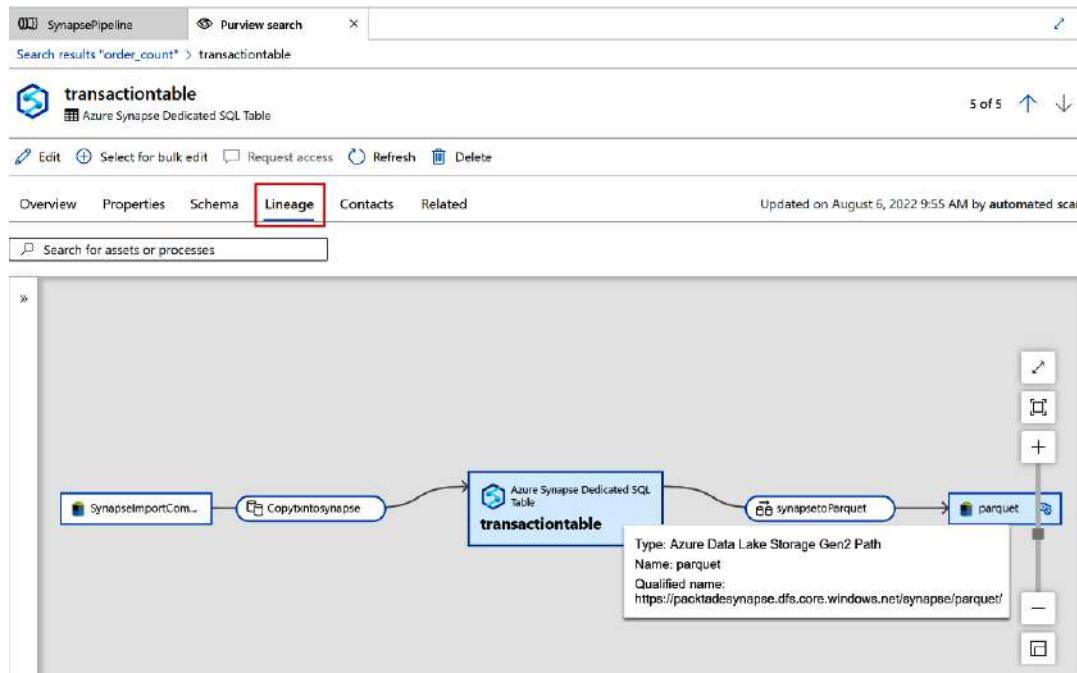


Figure 13.42 – Data lineage

If the Parquet file's destination is missing for you, go to the **synapsetoParquet** data flow, click on **source 1**, click on the **Source options** tab, and ensure **Input** is set to **Table**, as shown in the following screenshot. Make sure that you deselect the **Stored procedure** option for **Input**. Once set, publish the data flow, rerun the pipeline, and check the lineage after a few minutes:

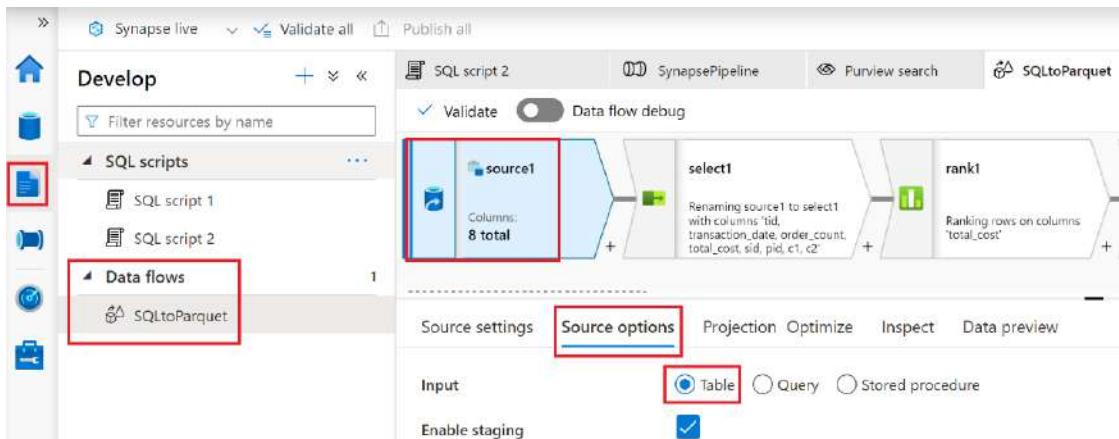


Figure 13.43 – Setting the data flow input

How it works...

Integrating a Synapse workspace with a Purview account helps us track data lineage easily. After each pipeline run, Synapse automatically updates the Purview account with the data lineage details. Data lineage for new pipelines and data flows is automatically updated to the Purview account. There is no need to perform a Purview scan via the Purview governance portal (as you did in the *Provisioning a Microsoft Purview account and creating a data catalog* recipe) to keep the lineage updated.

Applying Azure tags using PowerShell to multiple Azure resources

Azure tags help us organize the Azure resources we have created in Azure data engineering projects. Tags add key-value pairs to resources, which can help us easily categorize the resources and identify them. For example, adding a tag with the key set to `Environment` and the value set to `production` for resources can help us identify all production environment resources in our tenant. You can add multiple tags to the same resource too. For example, you may have two tags with `Environment` and `Project` as keys. `Environment` helps identify the environment that the resource belongs to, while `Project` helps identify which project the resource is part of.

It is always good practice to add tags at the time of resource creation. However, in some scenarios, you may need to add tags to several resources after they have been created. In this recipe, we will use a PowerShell script to tag several resources in one go.

Getting ready

You may perform this recipe in any Azure subscription with resources created.

How to do it...

In this recipe, we will add two tags – `Project` and `Classification` – to resources that match the following conditions:

- Resources starting with `Packt` – for example, resources named `packtadesqlpool`.
- All resources in resource groups whose names start with `Packt` – for example, all resources in a resource group named `PacktADESynapse`.

Follow these steps to add tags using PowerShell:

1. Connect to your Azure subscription using the `Connect-AzAccount` command. Copy and paste the following command into the PowerShell console and run it:

```
$Tags = @{"Project"="Packt"; "Classification"="Public"}
$str = get-AzResource | Where-Object {$_ .Name -like "packt*" -or $_ .ResourceGroupName -like "packt*"}
foreach ($Res in $str) {
    Set-AzResource -ResourceGroupName $Res.
    ResourceGroupName -Name $Res.Name -ResourceType $Res.
    ResourceType -Tag $tags -Force
}
```

This script will add the two tags with the following values:

- `Project` as the key and `Book` as the value. This implies that resources that match our condition belong to the project named `Book`.
- `Classification` as the key and `Public` as the value. This implies that resources that match our condition don't contain sensitive data and can be classified as public or non-sensitive.

The script will have a lengthy output. It should look similar to the following:

```
PS C:\Users\havenda>$Tags = @{"Project"="Packt"; "Classification"="Public"}
PS C:\Users\havenda>$str = get-AzResource | Where-Object {$_ .Name -like "packt*" -or $_ .ResourceGroupName -like "packt*"}
PS C:\Users\havenda>$foreach ($Res in $str) {
>>>     Set-AzResource -ResourceGroupName $Res.ResourceGroupName -Name $Res.Name -ResourceType $Res.ResourceType -Tag $tags -Force
}

Name          : PacktADEADF-vnet
ResourceId    : /subscriptions/[REDACTED]/resourceGroups/PacktADEADF/providers/Microsoft.Network/virtualNetworks/PacktADEADF-vnet
ResourceName   : PacktADEADF-vnet
ResourceType   : Microsoft.Network/virtualNetworks
ResourceGroupName: PacktADEADF
Location       : eastus
SubscriptionId: [REDACTED]
Tags          : {[Classification, Project], @(provisioningState=Succeeded; resourceGuid=fF1d8f3f-8292-4b3a-90cb-5e74c3a8ad8; addressSpace=; subnets=System.Object[]; virtualNetworkPeerings=System.Object[])
Properties    : {[enableDdosProtection=False]}
Etag          : E/192d9916-36f3-42f0-8af9-d495c291fad8

Name          : packtadesynapse
ResourceId    : /subscriptions/[REDACTED]/resourceGroups/PacktADESynapse/providers/Microsoft.Storage/storageAccounts/packtadesynapse
ResourceName   : packtadesynapse
ResourceType   : Microsoft.Storage/storageAccounts
Kind          : StorageV2
ResourceGroupName: PacktADESynapse
Location       : eastus
SubscriptionId: [REDACTED]
Tags          : {[Classification, Project], @(keyCreationTime=; privateEndpointConnections=System.Object[]; minimumTlsVersion=Tls1_2; allowBlobPublicAccess=True; isHnsEnabled=True; networkAcls=; supportsHttpsTrafficOnly=True; encryption=; accessTier=Hot; provisioningState=Succeeded; creationTime=2022-08-03T10:09:05.6287858Z; primaryEndpoints=; primaryLocation=eastus; statusOfPrimary=available; secondaryLocation=westus; statusOfSecondary=available; secondaryEndpoints=)}
Sku           : @(name=Standard RAGRS; tier=Standard)
```

Figure 12.44 – Publish Auto Pause Script

2. Let's use the Azure portal and explore the use of the Azure tags we've created. Go to the Azure portal and click on **All resources**. Then, click on **Add filter**. Notice that the tags we created are listed under **Tags**. Select **Classification** under the **Filter** option:

The screenshot shows the Azure portal's 'All resources' blade. At the top, there are several filter buttons: 'Subscription equals all', 'Resource group equals all', 'Type equals all', and 'Location equals all'. To the right of these is a red box around the '+ Add filter' button. A modal dialog titled 'Add filter' is open on the right. Inside the dialog, the 'Filter' dropdown is set to 'Classification'. Below it, the 'Operator' dropdown is set to 'Equals' and the 'Value' dropdown has 'all' selected. At the bottom of the dialog, there is an 'Apply' button and a checkbox for 'Select all'. Another checkbox, 'Public (26)', is checked and highlighted with a red box.

Figure 12.45 – Add filter

3. Select **Public** as the value and click the **Apply** button:

This screenshot shows the 'Add filter' dialog with the following settings: Filter set to 'Classification', Operator set to 'Equals', and Value set to 'all'. The 'Select all' checkbox is checked. The 'Public (26)' checkbox is also checked and highlighted with a red box. The 'Apply' button at the bottom left is also highlighted with a red box.

Figure 12.46 – Applying the filter

4. You will see that the resources we have tagged are listed:

The screenshot shows the Azure portal's 'All resources' view. At the top, there are several filter buttons: 'Create', 'Manage view', 'Refresh', 'Export to CSV', 'Open query', 'Assign tags', 'Delete', 'Filter for any field...', 'Subscription equals all', 'Resource group equals all', 'Type equals all', 'Location equals all', and 'Classification equals Public'. The 'Classification equals Public' button is highlighted with a red box. Below the filters, it says '5 Unsuspecting resources'. The main area lists 18 resources with columns for Name, Type, Resource group, Location, and Subscription. Most resources have a 'PacktADE' resource group and are located in 'East US' under 'Visual Studio Ultimate with MSDN'.

Name	Type	Resource group	Location	Subscription
SynapseAutoPause (azadeautomation/SynapseAutoPause)	Runbook	packtadesql	East US	Visual Studio Ultimate with MSDN
SQLWakeup (packtparcel/SQLWakeup)	Runbook	Packt	East US	Visual Studio Ultimate with MSDN
SelfHostedIR2_disk1_10647a2d3c7f44c28fb31ffe5032358b	Disk	PACKTADE	East US	Visual Studio Ultimate with MSDN
selfhostedir2537	Network Interface	PacktADE	East US	Visual Studio Ultimate with MSDN
SelfHostedIR2-nsg	Network security group	PacktADE	East US	Visual Studio Ultimate with MSDN
SelfHostedIR2-ip	Public IP address	PacktADE	East US	Visual Studio Ultimate with MSDN
SelfHostedIR2	Virtual machine	PacktADE	East US	Visual Studio Ultimate with MSDN
SelfHostedIR1_disk1_ce67427a0ec14819a3120bc157c0f1c7	Disk	PACKTADE	East US	Visual Studio Ultimate with MSDN
selfhostedir1310	Network Interface	PacktADE	East US	Visual Studio Ultimate with MSDN
SelfHostedIR1-nsg	Network security group	PacktADE	East US	Visual Studio Ultimate with MSDN
SelfHostedIR1-ip	Public IP address	PacktADE	East US	Visual Studio Ultimate with MSDN
SelfHostedIR1	Virtual machine	PacktADE	East US	Visual Studio Ultimate with MSDN
mscalesql (azadeautomation/mscalesql)	Runbook	packtadesql	East US	Visual Studio Ultimate with MSDN
packstoragepowershellv2	Storage account	Packtade-powershell	East US	Visual Studio Ultimate with MSDN
packtparcel	Automation Account	Packt	East US	Visual Studio Ultimate with MSDN
packtadesql\net	Virtual network	packtadesql	East US	Visual Studio Ultimate with MSDN

Figure 13.47 – Filtered tagged resources

How it works...

PowerShell script helps search through the subscription and apply the necessary tags. Let's look at what each command does:

- `get-Azresource | Where-Object {$_ .Name -like "packt*" -or $_ .ResourceGroupName -like "packt*"} :` This command filters for resources starting with Packt or that belong to resource groups starting with packt
- `foreach ($Res in $tr) :` This command loops through all the resources
- `Set-AzResource -ResourceGroupName $Res .ResourceGroupName -Name $Res .Name -ResourceType $Res .ResourceType -Tag $tags -Force :` This command applies the necessary tags

Even though they seem simple, Azure tags are an extremely powerful feature when you're managing complex projects that contain thousands of resources. For example, Azure tags are very useful in preparing Azure billing reports, where you need to find the monthly cost of the resources that belong to a particular project. Azure billing reports have a column containing tags that have been created, which we can use to filter (as we did in **All resources** in the Azure portal) and track the cost of a particular project easily.

Index

A

active geo-replication
configuring 204, 205
implementing, for Azure SQL
database with PowerShell 200
manual failover, performing to
secondary database 202, 203
readable secondary, creating 200, 201
removing 204
use cases 200
advanced workload management
workload groups, creating 436-442
alert
creating, to monitor Azure
storage account 45-54
Archive tier 16
Artificial Intelligence (AI) 247
Atomicity, Consistency, Isolation,
and Durability (ACID) 292
auto-failover group
creating 205-207
implementing, for Azure SQL
database with PowerShell 205
performing, to secondary server 208, 209
working 210

automatic tuning features
CREATE INDEX 241
DROP INDEX 241
FORCE PLAN 241
AutoResolveIntegrationRuntime 99
Azure blob
access tier, modifying 13, 14
copying, between containers 11, 12
deleting 15
downloading 15
lifecycle management, configuring for
blob objects with Azure portal 16-20
listing, in Azure storage container 13
managing in Azure Storage Account,
PowerShell used 10, 11, 16
Azure Blob storage
containers, creating with PowerShell 8-10
files, uploading with PowerShell 8-10
Azure Databricks clusters
creating 250-253
Azure Databricks environment
configuring 248, 249
Azure Databricks pools
creating 253-255
Azure Data Factory
pipeline, triggering 81-86

- provisioning 60-62
SSIS package, migrating to 135-149
- Azure Data Factory, components
activity 60
data flow 60
dataset 60
integration runtime 60
linked services 60
pipeline 60
- Azure Data Lake account
private links, configuring 28-34
- Azure Data Lake container
mounting, in Databricks 260-271
- Azure Key Vault
configuring, for Azure SQL
Database 171-177
used, for configuring encryption for
Azure Data Lake account 34-38
used, for integrating Databricks 255-259
- Azure Monitor
monitoring dashboards, building
for Synapse 473-482
- Azure portal
used, for configuring firewall for Azure
Data Lake account 22-24
used, for configuring virtual networks
for Azure Data Lake account 24-27
used, for monitoring Azure
SQL database 232
- Azure Resource Manager (ARM) 61
- Azure SQL Database
audit, configuring 242-246
automatic tuning 241
- Azure Key Vault, configuring 171-177
connecting, with PowerShell 152-155
creating 232, 233
- files, ingesting with Copy data activity 74-81
Hyperscale tier, configuring 192-197
- metrics, monitoring 233-235
monitoring, with Azure portal 232
monitoring, with diagnostic
settings 239-241
- private endpoints, configuring 163-171
provisioning, with PowerShell 152-155
- Query Performance Insight, using to find
resource-consuming queries 235-238
- used, for implementing active
geo-replication with PowerShell 200
- used, for implementing auto-failover
group with PowerShell 205
- used, for implementing vertical scaling
with PowerShell 219-232
- virtual network, configuring 163-171
- workload, executing 232, 233
- Azure SQL Database elastic pool
about 152
implementing, with PowerShell 156-163
- Azure storage account
provisioning, with Azure portal 2-6
provisioning, with PowerShell 6, 7
securing, with SAS using PowerShell 54-58
- Azure Synapse Analytics workspace
provisioning 312-315
- Azure Synapse SQL pool
result set caching, configuring 500-506
workload management,
implementing 430-435
- Azure tags
about 570
applying, with PowerShell to multiple
Azure resources 570-573

B

- blobs
securing, with SAS 55, 56

Blob storage accounts
accessing, with managed identities 39-45

C

Comma-Separated Value (CSV) file 271

compute nodes 484

container

securing, with SAS 57

continuous integration and continuous deployment (CI/CD) 61

control node 484

Cool tier 16

Copy activity

adding, to ingest file to Azure

SQL Database 74-81

used, for copying files to database from data lake 62-64

Copy data wizard

used, for copying data from SQL Server virtual machine to data lake 87-97

COPY INTO

used, for loading data into dedicated SQL pool 408-413

Create Table As (CTAS) 408

current partitioning 383

D

data

archiving, with partitioned tables 422-430

copying, with Synapse data flow 352-364

loading, into dedicated SQL pools with PolyBase using T-SQL 400-408

loading, into dedicated SQL pool

using **COPY INTO** 408-413

processing, with lake database 325-331

processing, with Spark pools 325-331

provisioning, with serverless

SQL pool 315-320

querying, in lake database from serverless SQL pool 332-336

visualizing, with Power BI by connecting to serverless SQL pool 345-349

database throughput units (DTUs) 156

Databricks

Azure Data Lake container, mounting 260-271

integrating, with Azure Key Vault 255-259

Databricks Delta Lake table

connecting, to Power BI 300-310

Databricks File System (DBFS) storage 281

Databricks notebook

scheduling, with job clusters 282-292

used, for processing data 271-281

data catalog

creating 549-564

Data Definition Language (DDL) statement 499

data flows

monitoring 375-378

optimization, by configuring partitions 379-383

schema changes dynamically, handling with schema drift 389-398

data lake

data, copying with Copy data wizard from SQL Server virtual machine to 87-97

data lineage

tracking, with Microsoft Purview 564-570

data pipelines

monitoring 375-378

data transformation activities

performing, with filter

transformation 364-373

performing, with join
 transformation 364-373

performing, with sort
 transformation 364-373

Datawarehouse Units (DWUs) 474, 484

dedicated SQL pool

- COPY INTO, used for loading data
 - into dedicated SQL pool 408-413
- T-SQL, used for loading data
 - with PolyBase 400-408

Delta Lake tables

- working with 292-300

Delta tables

- optimizing, in Synapse Spark Pool
 - lake database 522-528

distributed denial-of-service (DDOS) 163

Distributed File Systems (DFS) 30

distributed tables

- creating 413-416

Domain Name System (DNS) 30

DTU-based model 219

Dynamic Management Views (DMVs)

- about 467
- data skew, monitoring 467-470
- index health, monitoring 470, 472

E

encryption

- configuring, with Azure Key Vault for
 - Azure Data Lake account 34-39

external data source 335

external file format 335

external table 335

Extract, Transform, and Load (ETL) 135, 247, 389, 505

F

files

- uploading, to Azure Data Lake
 - with PowerShell 8-10

file share 2

Filter activity

- current date files, filtering with 71-73

filter transformation

- about 364
- used, for performing data transformation
 - activities 364-375

firewall

- configuring, for Azure Data Lake
 - account with Azure portal 22-24

ForEach activity

- adding, to loop through files 73, 74

Fully Qualified Domain Name (FQDN) 31

G

Get Metadata activity

- using, to obtain filenames 68-70

H

hash distribution 491

high availability

- configuring, for self-hosted IR 120-128
- configuring, to Hyperscale tier of
 - Azure SQL Database 211-218

Hot tier 16

Hyperscale tier of Azure SQL Database

- configuring 192-197
- geo-replica 211
- high availability, configuring to 211-218
- high-availability replica 211
- named replica 211

I

Integration Runtime (IR) 99

J

job clusters

- used, for scheduling Databricks notebook 282-292

join transformation

- about 364
- used, for performing data transformation activities 364-375

K

Kusto queries

- using, to monitor SQL and Spark pools 451-459

Kusto Query Language (KQL) 244

L

lake database

- used, for processing data 325-331

linked service

- creating 64-66

Log Analytics workbook

- used, for monitoring Synapse integration pipelines 536-542

Log Analytics workspace

- about 444

- configuring, for Synapse Spark pools 448-451

- configuring, for Synapse SQL pools 444-447

log categories

- DmsWorkers 448

- ExecRequests 447

RequestSteps 447

SqlRequests 447

Waits 448

longer backup retention

- configuring, for Synapse SQL database 506-514

M

Machine Learning (ML) 247

managed identity

- about 507

- used, for accessing Blob storage accounts 39-45

managed instances 152

Microsoft IR 113

Microsoft Purview

- about 549

- used, for integrating Synapse workspace 564-570

- used, for tracking data lineage 564-570

Microsoft Purview account

- provisioning 549-564

monitoring dashboards, for Synapse

- building, with Azure Monitor 473-482

N

notebooks

- scheduling, to process data incrementally 336-344

O

optimization techniques, Delta

- lake query performance

- reference link 534

OPTIMIZE command 527, 528

P

Parquet sink transformation 394
partitioned tables
 used, for archiving data 422-430
 used, for creating partitions 422-430
partitions
 about 528
 configuring, to optimize data flows 379-383
 creating, with partitioned tables 422-430

PolyBase
 T-SQL, used for data loading into
 dedicated SQL pools with 400-408

Power BI
 about 345
 Databricks Delta Lake table,
 connecting to 300-310
 used, for visualizing data by connecting
 to serverless SQL pool 345-349

PowerShell
 active geo-replication, implementing
 for Azure SQL database 200
 auto-failover group, implementing
 for Azure SQL database 205
 used, for applying Azure tags to multiple
 Azure resources 570-573
 used, for connecting to Azure
 SQL Database 152-155
 used, for creating containers to
 Azure Data Lake 8-10
 used, for implementing Azure SQL
 Database elastic pool 156-163
 used, for provisioning Azure
 SQL Database 152-155
 used, for provisioning Azure
 storage account 6, 7
 used, for securing Azure storage
 account with SAS 54-58

used, for uploading files to
 Azure Data Lake 8-10
vertical scaling, implementing for
 Azure SQL database 219-232

private endpoints
 configuring, for Azure SQL
 Database 163-171

private links
 configuring, for Azure Data
 Lake account 28-34

Q

query performance
 optimizing, in Synapse Spark Pool 528-534
Query Performance Insight 235
query plan
 analyzing 485-493
queue storage 2

R

replication table cache
 monitoring 493-500
 rebuilding 493-500
replication tables 493
result set caching
 about 500
 configuring, in Azure Synapse
 SQL Pool 500-506

S

schema drift
 used, for handling schema changes
 dynamically in data flows 389-398
self-hosted IR
 about 120, 128

-
- automatic patch update, disabling 129-135
 - configuring 100-113
 - high availability, configuring 120-128
 - latest patches, installing 129-135
 - patching 128
 - serverless SQL database
 - wake-up script, configuring 177-192
 - wake-up script, provisioning 177-192
 - serverless SQL pool
 - data, querying in lake database 332-336
 - data, visualizing with Power BI by connecting to 345-349
 - used, for analyzing data 315
 - used, for data analyzing 316-320
 - Shared Access Signature (SAS)
 - used, for securing blobs 55, 56
 - used, for securing container 57
 - shared self-hosted IR
 - configuring 113-119
 - sort transformation
 - about 364
 - used, for performing data transformation activities 364-375
 - Spark pools
 - configuring 322-325
 - Kusto queries, using to monitor 451-459
 - provisioning 322-325
 - used, for processing data 325-331
 - SQLCMD utility
 - reference link 155
 - SQL injection 163
 - SQL pool
 - Kusto queries, using to monitor 451-459
 - SQL queries
 - tracing, for dedicated SQL pool to Synapse integration pipelines 543-549
 - SQL Server Integration Services (SSIS) 99, 135
 - SQL Server Management Studio (SSMS) 155, 211
 - SQL Server virtual machine
 - data, copying with Copy data wizard to data lake from 87-97
 - SSIS package
 - migrating, to Azure Data Factory 135-149
 - standalone Azure SQL Database 152
 - statistics
 - creating 417-422
 - update, automating 417-422
 - Synapse data flow
 - parameterizing 383-388
 - used, for copying data 352-364
 - Synapse dedicated SQL pool
 - architecture 484, 485
 - options 413
 - Synapse integration pipelines
 - monitoring, with Log Analytics and workbooks 536-542
 - Synapse Spark Pool lake database
 - Delta tables, optimizing 522-528
 - Synapse Spark pools
 - Log Analytics workspace, configuring for 448-451
 - query performance, optimizing 528-534
 - Synapse SQL database
 - longer backup retention, configuring for 506-514
 - Synapse SQL pool
 - about 484
 - auto pausing 514-522
 - Log Analytics workspace, configuring for 444-447
 - Synapse workspace
 - integrating, with Microsoft Purview 564-570

T

table distribution
fixing 485-493
modifying 413-416
table storage 2
T-SQL
used, for loading data into dedicated
SQL pools with PolyBase 400-408

U

unsupported scenarios, result set caching
reference link 506
User Acceptance Test (UAT) 128, 522

V

VACUUM command 527, 528
vCore-based model 219
vertical scaling
implementing, for Azure SQL database
with PowerShell 219-232
virtual machine (VM) 155
virtual network
configuring, for Azure Data Lake
account with Azure portal 24-27
configuring, for Azure SQL
Database 163-171
creating 164

W

wake-up script
configuring, for serverless SQL
database 177-192
provisioning, for serverless SQL
database 177-192

workbooks
about 459
creating, in Log Analytics workspace to
visualize monitoring data 459-466
used, for monitoring Synapse
integration pipelines 536, 537
workload groups
creating, for advanced workload
management 436-442
workload management
implementing, in Azure Synapse
SQL pool 430-435

Z

Z-Ordering 528, 533, 534



Packt . com

Subscribe to our online digital library for full access to over 7,000 books and videos, as well as industry leading tools to help you plan your personal development and advance your career. For more information, please visit our website.

Why subscribe?

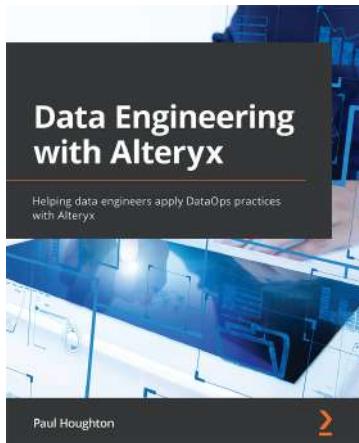
- Spend less time learning and more time coding with practical eBooks and Videos from over 4,000 industry professionals
- Improve your learning with Skill Plans built especially for you
- Get a free eBook or video every month
- Fully searchable for easy access to vital information
- Copy and paste, print, and bookmark content

Did you know that Packt offers eBook versions of every book published, with PDF and ePub files available? You can upgrade to the eBook version at packt.com and as a print book customer, you are entitled to a discount on the eBook copy. Get in touch with us at customercare@packtpub.com for more details.

At www.packt.com, 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 Packt books and eBooks.

Other Books You May Enjoy

If you enjoyed this book, you may be interested in these other books by Packt:

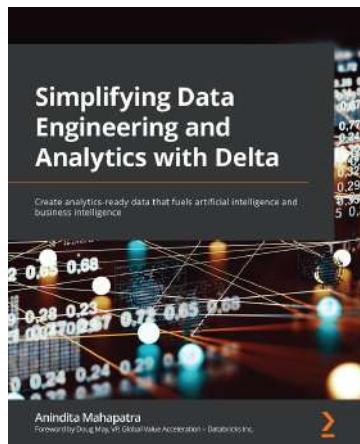


Data Engineering with Alteryx

Paul Houghton

ISBN: 9781803236483

- Build a working pipeline to integrate an external data source
- Develop monitoring processes for the pipeline example
- Understand and apply DataOps principles to an Alteryx data pipeline
- Gain skills for data engineering with the Alteryx software stack
- Work with spatial analytics and machine learning techniques in an Alteryx workflow
- Explore Alteryx workflow deployment strategies using metadata validation and continuous integration
- Organize content on Alteryx Server and secure user access



Simplifying Data Engineering and Analytics with Delta

Anindita Mahapatra

ISBN: 9781801814867

- Explore the key challenges of traditional data lakes
 - Appreciate the unique features of Delta that come out of the box
 - Address reliability, performance, and governance concerns using Delta
 - Analyze the open data format for an extensible and pluggable architecture
 - Handle multiple use cases to support BI, AI, streaming, and data discovery
 - Discover how common data and machine learning design patterns are executed on Delta
 - Build and deploy data and machine learning pipelines at scale using Delta

Packt is searching for authors like you

If you're interested in becoming an author for Packt, please visit authors.packtpub.com and apply today. We have worked with thousands of developers and tech professionals, just like you, to help them share their insight with the global tech community. You can make a general application, apply for a specific hot topic that we are recruiting an author for, or submit your own idea.

Share your thoughts

Now you've finished *Azure Data Engineering Cookbook, Second Edition*, we'd love to hear your thoughts! If you purchased the book from Amazon, please click [here](#) to go straight to the Amazon review page for this book and share your feedback or leave a review on the site that you purchased it from.

Your review is important to us and the tech community and will help us make sure we're delivering excellent quality content.

