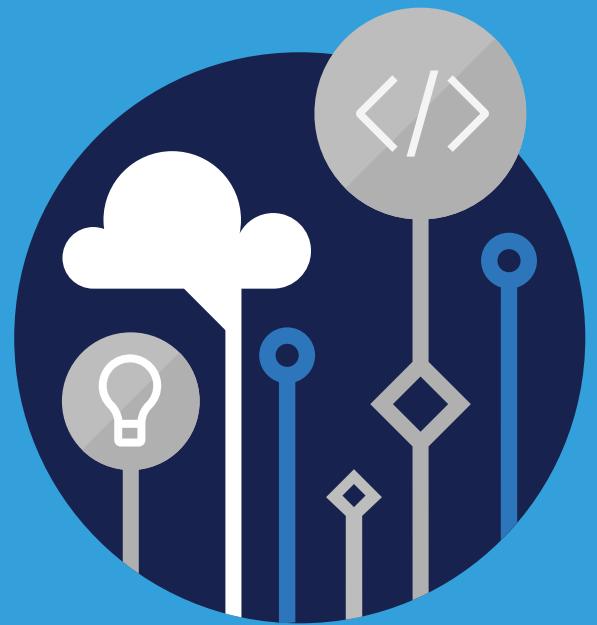


Microsoft
Official
Course



DP-203T00

Data Engineering on
Microsoft Azure

DP-203T00
Data Engineering on Microsoft
Azure

II Disclaimer

Information in this document, including URL and other Internet Web site references, is subject to change without notice. Unless otherwise noted, the example companies, organizations, products, domain names, e-mail addresses, logos, people, places, and events depicted herein are fictitious, and no association with any real company, organization, product, domain name, e-mail address, logo, person, place or event is intended or should be inferred. Complying with all applicable copyright laws is the responsibility of the user. Without limiting the rights under copyright, no part of this document may be reproduced, stored in or introduced into a retrieval system, or transmitted in any form or by any means (electronic, mechanical, photocopying, recording, or otherwise), or for any purpose, without the express written permission of Microsoft Corporation.

Microsoft may have patents, patent applications, trademarks, copyrights, or other intellectual property rights covering subject matter in this document. Except as expressly provided in any written license agreement from Microsoft, the furnishing of this document does not give you any license to these patents, trademarks, copyrights, or other intellectual property.

The names of manufacturers, products, or URLs are provided for informational purposes only and Microsoft makes no representations and warranties, either expressed, implied, or statutory, regarding these manufacturers or the use of the products with any Microsoft technologies. The inclusion of a manufacturer or product does not imply endorsement of Microsoft of the manufacturer or product. Links may be provided to third party sites. Such sites are not under the control of Microsoft and Microsoft is not responsible for the contents of any linked site or any link contained in a linked site, or any changes or updates to such sites. Microsoft is not responsible for webcasting or any other form of transmission received from any linked site. Microsoft is providing these links to you only as a convenience, and the inclusion of any link does not imply endorsement of Microsoft of the site or the products contained therein.

© 2019 Microsoft Corporation. All rights reserved.

Microsoft and the trademarks listed at <http://www.microsoft.com/trademarks>¹ are trademarks of the Microsoft group of companies. All other trademarks are property of their respective owners.

¹ <http://www.microsoft.com/trademarks>

MICROSOFT LICENSE TERMS

MICROSOFT INSTRUCTOR-LED COURSEWARE

These license terms are an agreement between Microsoft Corporation (or based on where you live, one of its affiliates) and you. Please read them. They apply to your use of the content accompanying this agreement which includes the media on which you received it, if any. These license terms also apply to Trainer Content and any updates and supplements for the Licensed Content unless other terms accompany those items. If so, those terms apply.

**BY ACCESSING, DOWNLOADING OR USING THE LICENSED CONTENT, YOU ACCEPT THESE TERMS.
IF YOU DO NOT ACCEPT THEM, DO NOT ACCESS, DOWNLOAD OR USE THE LICENSED CONTENT.**

If you comply with these license terms, you have the rights below for each license you acquire.

1. DEFINITIONS.

1. "Authorized Learning Center" means a Microsoft Imagine Academy (MSIA) Program Member, Microsoft Learning Competency Member, or such other entity as Microsoft may designate from time to time.
2. "Authorized Training Session" means the instructor-led training class using Microsoft Instructor-Led Courseware conducted by a Trainer at or through an Authorized Learning Center.
3. "Classroom Device" means one (1) dedicated, secure computer that an Authorized Learning Center owns or controls that is located at an Authorized Learning Center's training facilities that meets or exceeds the hardware level specified for the particular Microsoft Instructor-Led Courseware.
4. "End User" means an individual who is (i) duly enrolled in and attending an Authorized Training Session or Private Training Session, (ii) an employee of an MPN Member (defined below), or (iii) a Microsoft full-time employee, a Microsoft Imagine Academy (MSIA) Program Member, or a Microsoft Learn for Educators – Validated Educator.
5. "Licensed Content" means the content accompanying this agreement which may include the Microsoft Instructor-Led Courseware or Trainer Content.
6. "Microsoft Certified Trainer" or "MCT" means an individual who is (i) engaged to teach a training session to End Users on behalf of an Authorized Learning Center or MPN Member, and (ii) currently certified as a Microsoft Certified Trainer under the Microsoft Certification Program.
7. "Microsoft Instructor-Led Courseware" means the Microsoft-branded instructor-led training course that educates IT professionals, developers, students at an academic institution, and other learners on Microsoft technologies. A Microsoft Instructor-Led Courseware title may be branded as MOC, Microsoft Dynamics, or Microsoft Business Group courseware.
8. "Microsoft Imagine Academy (MSIA) Program Member" means an active member of the Microsoft Imagine Academy Program.
9. "Microsoft Learn for Educators – Validated Educator" means an educator who has been validated through the Microsoft Learn for Educators program as an active educator at a college, university, community college, polytechnic or K-12 institution.
10. "Microsoft Learning Competency Member" means an active member of the Microsoft Partner Network program in good standing that currently holds the Learning Competency status.
11. "MOC" means the "Official Microsoft Learning Product" instructor-led courseware known as Microsoft Official Course that educates IT professionals, developers, students at an academic institution, and other learners on Microsoft technologies.
12. "MPN Member" means an active Microsoft Partner Network program member in good standing.

13. "Personal Device" means one (1) personal computer, device, workstation or other digital electronic device that you personally own or control that meets or exceeds the hardware level specified for the particular Microsoft Instructor-Led Courseware.
 14. "Private Training Session" means the instructor-led training classes provided by MPN Members for corporate customers to teach a predefined learning objective using Microsoft Instructor-Led Courseware. These classes are not advertised or promoted to the general public and class attendance is restricted to individuals employed by or contracted by the corporate customer.
 15. "Trainer" means (i) an academically accredited educator engaged by a Microsoft Imagine Academy Program Member to teach an Authorized Training Session, (ii) an academically accredited educator validated as a Microsoft Learn for Educators – Validated Educator, and/or (iii) a MCT.
 16. "Trainer Content" means the trainer version of the Microsoft Instructor-Led Courseware and additional supplemental content designated solely for Trainers' use to teach a training session using the Microsoft Instructor-Led Courseware. Trainer Content may include Microsoft PowerPoint presentations, trainer preparation guide, train the trainer materials, Microsoft One Note packs, classroom setup guide and Pre-release course feedback form. To clarify, Trainer Content does not include any software, virtual hard disks or virtual machines.
2. **USE RIGHTS.** The Licensed Content is licensed, not sold. The Licensed Content is licensed on a **one copy per user basis**, such that you must acquire a license for each individual that accesses or uses the Licensed Content.
- 2.1 Below are five separate sets of use rights. Only one set of rights apply to you.
 1. **If you are a Microsoft Imagine Academy (MSIA) Program Member:**
 1. Each license acquired on behalf of yourself may only be used to review one (1) copy of the Microsoft Instructor-Led Courseware in the form provided to you. If the Microsoft Instructor-Led Courseware is in digital format, you may install one (1) copy on up to three (3) Personal Devices. You may not install the Microsoft Instructor-Led Courseware on a device you do not own or control.
 2. For each license you acquire on behalf of an End User or Trainer, you may either:
 1. distribute one (1) hard copy version of the Microsoft Instructor-Led Courseware to one (1) End User who is enrolled in the Authorized Training Session, and only immediately prior to the commencement of the Authorized Training Session that is the subject matter of the Microsoft Instructor-Led Courseware being provided, **or**
 2. provide one (1) End User with the unique redemption code and instructions on how they can access one (1) digital version of the Microsoft Instructor-Led Courseware, **or**
 3. provide one (1) Trainer with the unique redemption code and instructions on how they can access one (1) Trainer Content.
 3. For each license you acquire, you must comply with the following:
 1. you will only provide access to the Licensed Content to those individuals who have acquired a valid license to the Licensed Content,
 2. you will ensure each End User attending an Authorized Training Session has their own valid licensed copy of the Microsoft Instructor-Led Courseware that is the subject of the Authorized Training Session,
 3. you will ensure that each End User provided with the hard-copy version of the Microsoft Instructor-Led Courseware will be presented with a copy of this agreement and each End

User will agree that their use of the Microsoft Instructor-Led Courseware will be subject to the terms in this agreement prior to providing them with the Microsoft Instructor-Led Courseware. Each individual will be required to denote their acceptance of this agreement in a manner that is enforceable under local law prior to their accessing the Microsoft Instructor-Led Courseware,

4. you will ensure that each Trainer teaching an Authorized Training Session has their own valid licensed copy of the Trainer Content that is the subject of the Authorized Training Session,
5. you will only use qualified Trainers who have in-depth knowledge of and experience with the Microsoft technology that is the subject of the Microsoft Instructor-Led Courseware being taught for all your Authorized Training Sessions,
6. you will only deliver a maximum of 15 hours of training per week for each Authorized Training Session that uses a MOC title, and
7. you acknowledge that Trainers that are not MCTs will not have access to all of the trainer resources for the Microsoft Instructor-Led Courseware.

2. If you are a Microsoft Learning Competency Member:

1. Each license acquire may only be used to review one (1) copy of the Microsoft Instructor-Led Courseware in the form provided to you. If the Microsoft Instructor-Led Courseware is in digital format, you may install one (1) copy on up to three (3) Personal Devices. You may not install the Microsoft Instructor-Led Courseware on a device you do not own or control.
2. For each license you acquire on behalf of an End User or MCT, you may either:
 1. distribute one (1) hard copy version of the Microsoft Instructor-Led Courseware to one (1) End User attending the Authorized Training Session and only immediately prior to the commencement of the Authorized Training Session that is the subject matter of the Microsoft Instructor-Led Courseware provided, **or**
 2. provide one (1) End User attending the Authorized Training Session with the unique redemption code and instructions on how they can access one (1) digital version of the Microsoft Instructor-Led Courseware, **or**
 3. you will provide one (1) MCT with the unique redemption code and instructions on how they can access one (1) Trainer Content.
3. For each license you acquire, you must comply with the following:
 1. you will only provide access to the Licensed Content to those individuals who have acquired a valid license to the Licensed Content,
 2. you will ensure that each End User attending an Authorized Training Session has their own valid licensed copy of the Microsoft Instructor-Led Courseware that is the subject of the Authorized Training Session,
 3. you will ensure that each End User provided with a hard-copy version of the Microsoft Instructor-Led Courseware will be presented with a copy of this agreement and each End User will agree that their use of the Microsoft Instructor-Led Courseware will be subject to the terms in this agreement prior to providing them with the Microsoft Instructor-Led Courseware. Each individual will be required to denote their acceptance of this agreement in a manner that is enforceable under local law prior to their accessing the Microsoft Instructor-Led Courseware,

4. you will ensure that each MCT teaching an Authorized Training Session has their own valid licensed copy of the Trainer Content that is the subject of the Authorized Training Session,
5. you will only use qualified MCTs who also hold the applicable Microsoft Certification credential that is the subject of the MOC title being taught for all your Authorized Training Sessions using MOC,
6. you will only provide access to the Microsoft Instructor-Led Courseware to End Users, and
7. you will only provide access to the Trainer Content to MCTs.

3. If you are a MPN Member:

1. Each license acquired on behalf of yourself may only be used to review one (1) copy of the Microsoft Instructor-Led Courseware in the form provided to you. If the Microsoft Instructor-Led Courseware is in digital format, you may install one (1) copy on up to three (3) Personal Devices. You may not install the Microsoft Instructor-Led Courseware on a device you do not own or control.
2. For each license you acquire on behalf of an End User or Trainer, you may either:
 1. distribute one (1) hard copy version of the Microsoft Instructor-Led Courseware to one (1) End User attending the Private Training Session, and only immediately prior to the commencement of the Private Training Session that is the subject matter of the Microsoft Instructor-Led Courseware being provided, **or**
 2. provide one (1) End User who is attending the Private Training Session with the unique redemption code and instructions on how they can access one (1) digital version of the Microsoft Instructor-Led Courseware, **or**
 3. you will provide one (1) Trainer who is teaching the Private Training Session with the unique redemption code and instructions on how they can access one (1) Trainer Content.
3. For each license you acquire, you must comply with the following:
 1. you will only provide access to the Licensed Content to those individuals who have acquired a valid license to the Licensed Content,
 2. you will ensure that each End User attending an Private Training Session has their own valid licensed copy of the Microsoft Instructor-Led Courseware that is the subject of the Private Training Session,
 3. you will ensure that each End User provided with a hard copy version of the Microsoft Instructor-Led Courseware will be presented with a copy of this agreement and each End User will agree that their use of the Microsoft Instructor-Led Courseware will be subject to the terms in this agreement prior to providing them with the Microsoft Instructor-Led Courseware. Each individual will be required to denote their acceptance of this agreement in a manner that is enforceable under local law prior to their accessing the Microsoft Instructor-Led Courseware,
 4. you will ensure that each Trainer teaching an Private Training Session has their own valid licensed copy of the Trainer Content that is the subject of the Private Training Session,

5. you will only use qualified Trainers who hold the applicable Microsoft Certification credential that is the subject of the Microsoft Instructor-Led Courseware being taught for all your Private Training Sessions,
6. you will only use qualified MCTs who hold the applicable Microsoft Certification credential that is the subject of the MOC title being taught for all your Private Training Sessions using MOC,
7. you will only provide access to the Microsoft Instructor-Led Courseware to End Users, and
8. you will only provide access to the Trainer Content to Trainers.

4. If you are an End User:

For each license you acquire, you may use the Microsoft Instructor-Led Courseware solely for your personal training use. If the Microsoft Instructor-Led Courseware is in digital format, you may access the Microsoft Instructor-Led Courseware online using the unique redemption code provided to you by the training provider and install and use one (1) copy of the Microsoft Instructor-Led Courseware on up to three (3) Personal Devices. You may also print one (1) copy of the Microsoft Instructor-Led Courseware. You may not install the Microsoft Instructor-Led Courseware on a device you do not own or control.

5. If you are a Trainer.

1. For each license you acquire, you may install and use one (1) copy of the Trainer Content in the form provided to you on one (1) Personal Device solely to prepare and deliver an Authorized Training Session or Private Training Session, and install one (1) additional copy on another Personal Device as a backup copy, which may be used only to reinstall the Trainer Content. You may not install or use a copy of the Trainer Content on a device you do not own or control. You may also print one (1) copy of the Trainer Content solely to prepare for and deliver an Authorized Training Session or Private Training Session.

2. If you are an MCT, you may customize the written portions of the Trainer Content that are logically associated with instruction of a training session in accordance with the most recent version of the MCT agreement.
3. If you elect to exercise the foregoing rights, you agree to comply with the following: (i) customizations may only be used for teaching Authorized Training Sessions and Private Training Sessions, and (ii) all customizations will comply with this agreement. For clarity, any use of "customize" refers only to changing the order of slides and content, and/or not using all the slides or content, it does not mean changing or modifying any slide or content.

- 2.2 **Separation of Components.** The Licensed Content is licensed as a single unit and you may not separate their components and install them on different devices.
- 2.3 **Redistribution of Licensed Content.** Except as expressly provided in the use rights above, you may not distribute any Licensed Content or any portion thereof (including any permitted modifications) to any third parties without the express written permission of Microsoft.
- 2.4 **Third Party Notices.** The Licensed Content may include third party code that Microsoft, not the third party, licenses to you under this agreement. Notices, if any, for the third party code are included for your information only.
- 2.5 **Additional Terms.** Some Licensed Content may contain components with additional terms, conditions, and licenses regarding its use. Any non-conflicting terms in those conditions and licenses also apply to your use of that respective component and supplements the terms described in this agreement.

3. **LICENSED CONTENT BASED ON PRE-RELEASE TECHNOLOGY.** If the Licensed Content's subject matter is based on a pre-release version of Microsoft technology ("Pre-release"), then in addition to the other provisions in this agreement, these terms also apply:
 1. **Pre-Release Licensed Content.** This Licensed Content subject matter is on the Pre-release version of the Microsoft technology. The technology may not work the way a final version of the technology will and we may change the technology for the final version. We also may not release a final version. Licensed Content based on the final version of the technology may not contain the same information as the Licensed Content based on the Pre-release version. Microsoft is under no obligation to provide you with any further content, including any Licensed Content based on the final version of the technology.
 2. **Feedback.** If you agree to give feedback about the Licensed Content to Microsoft, either directly or through its third party designee, you give to Microsoft without charge, the right to use, share and commercialize your feedback in any way and for any purpose. You also give to third parties, without charge, any patent rights needed for their products, technologies and services to use or interface with any specific parts of a Microsoft technology, Microsoft product, or service that includes the feedback. You will not give feedback that is subject to a license that requires Microsoft to license its technology, technologies, or products to third parties because we include your feedback in them. These rights survive this agreement.
 3. **Pre-release Term.** If you are an Microsoft Imagine Academy Program Member, Microsoft Learning Competency Member, MPN Member, Microsoft Learn for Educators – Validated Educator, or Trainer, you will cease using all copies of the Licensed Content on the Pre-release technology upon (i) the date which Microsoft informs you is the end date for using the Licensed Content on the Pre-release technology, or (ii) sixty (60) days after the commercial release of the technology that is the subject of the Licensed Content, whichever is earliest ("Pre-release term"). Upon expiration or termination of the Pre-release term, you will irretrievably delete and destroy all copies of the Licensed Content in your possession or under your control.
 4. **SCOPE OF LICENSE.** The Licensed Content is licensed, not sold. This agreement only gives you some rights to use the Licensed Content. Microsoft reserves all other rights. Unless applicable law gives you more rights despite this limitation, you may use the Licensed Content only as expressly permitted in this agreement. In doing so, you must comply with any technical limitations in the Licensed Content that only allows you to use it in certain ways. Except as expressly permitted in this agreement, you may not:
 - access or allow any individual to access the Licensed Content if they have not acquired a valid license for the Licensed Content,
 - alter, remove or obscure any copyright or other protective notices (including watermarks), branding or identifications contained in the Licensed Content,
 - modify or create a derivative work of any Licensed Content,
 - publicly display, or make the Licensed Content available for others to access or use,
 - copy, print, install, sell, publish, transmit, lend, adapt, reuse, link to or post, make available or distribute the Licensed Content to any third party,
 - work around any technical limitations in the Licensed Content, or
 - reverse engineer, decompile, remove or otherwise thwart any protections or disassemble the Licensed Content except and only to the extent that applicable law expressly permits, despite this limitation.
 5. **RESERVATION OF RIGHTS AND OWNERSHIP.** Microsoft reserves all rights not expressly granted to you in this agreement. The Licensed Content is protected by copyright and other intellectual property

laws and treaties. Microsoft or its suppliers own the title, copyright, and other intellectual property rights in the Licensed Content.

6. **EXPORT RESTRICTIONS.** The Licensed Content is subject to United States export laws and regulations. You must comply with all domestic and international export laws and regulations that apply to the Licensed Content. These laws include restrictions on destinations, end users and end use. For additional information, see www.microsoft.com/exporting.
7. **SUPPORT SERVICES.** Because the Licensed Content is provided "as is", we are not obligated to provide support services for it.
8. **TERMINATION.** Without prejudice to any other rights, Microsoft may terminate this agreement if you fail to comply with the terms and conditions of this agreement. Upon termination of this agreement for any reason, you will immediately stop all use of and delete and destroy all copies of the Licensed Content in your possession or under your control.
9. **LINKS TO THIRD PARTY SITES.** You may link to third party sites through the use of the Licensed Content. The third party sites are not under the control of Microsoft, and Microsoft is not responsible for the contents of any third party sites, any links contained in third party sites, or any changes or updates to third party sites. Microsoft is not responsible for webcasting or any other form of transmission received from any third party sites. Microsoft is providing these links to third party sites to you only as a convenience, and the inclusion of any link does not imply an endorsement by Microsoft of the third party site.
10. **ENTIRE AGREEMENT.** This agreement, and any additional terms for the Trainer Content, updates and supplements are the entire agreement for the Licensed Content, updates and supplements.
11. **APPLICABLE LAW.**
 1. United States. If you acquired the Licensed Content in the United States, Washington state law governs the interpretation of this agreement and applies to claims for breach of it, regardless of conflict of laws principles. The laws of the state where you live govern all other claims, including claims under state consumer protection laws, unfair competition laws, and in tort.
 2. Outside the United States. If you acquired the Licensed Content in any other country, the laws of that country apply.
12. **LEGAL EFFECT.** This agreement describes certain legal rights. You may have other rights under the laws of your country. You may also have rights with respect to the party from whom you acquired the Licensed Content. This agreement does not change your rights under the laws of your country if the laws of your country do not permit it to do so.
13. **DISCLAIMER OF WARRANTY. THE LICENSED CONTENT IS LICENSED "AS-IS" AND "AS AVAILABLE." YOU BEAR THE RISK OF USING IT. MICROSOFT AND ITS RESPECTIVE AFFILIATES GIVES NO EXPRESS WARRANTIES, GUARANTEES, OR CONDITIONS. YOU MAY HAVE ADDITIONAL CONSUMER RIGHTS UNDER YOUR LOCAL LAWS WHICH THIS AGREEMENT CANNOT CHANGE. TO THE EXTENT PERMITTED UNDER YOUR LOCAL LAWS, MICROSOFT AND ITS RESPECTIVE AFFILIATES EXCLUDES ANY IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NON-INFRINGEMENT.**
14. **LIMITATION ON AND EXCLUSION OF REMEDIES AND DAMAGES. YOU CAN RECOVER FROM MICROSOFT, ITS RESPECTIVE AFFILIATES AND ITS SUPPLIERS ONLY DIRECT DAMAGES UP TO US\$5.00. YOU CANNOT RECOVER ANY OTHER DAMAGES, INCLUDING CONSEQUENTIAL, LOST PROFITS, SPECIAL, INDIRECT OR INCIDENTAL DAMAGES.**

This limitation applies to

- anything related to the Licensed Content, services, content (including code) on third party Internet sites or third-party programs; and
- claims for breach of contract, breach of warranty, guarantee or condition, strict liability, negligence, or other tort to the extent permitted by applicable law.

It also applies even if Microsoft knew or should have known about the possibility of the damages. The above limitation or exclusion may not apply to you because your country may not allow the exclusion or limitation of incidental, consequential, or other damages.

Please note: As this Licensed Content is distributed in Quebec, Canada, some of the clauses in this agreement are provided below in French.

Remarque : Ce le contenu sous licence étant distribué au Québec, Canada, certaines des clauses dans ce contrat sont fournies ci-dessous en français.

EXONÉRATION DE GARANTIE. Le contenu sous licence visé par une licence est offert « tel quel ». Toute utilisation de ce contenu sous licence est à votre seule risque et péril. Microsoft n'accorde aucune autre garantie expresse. Vous pouvez bénéficier de droits additionnels en vertu du droit local sur la protection dues consommateurs, que ce contrat ne peut modifier. La ou elles sont permises par le droit locale, les garanties implicites de qualité marchande, d'adéquation à un usage particulier et d'absence de contrefaçon sont exclues.

LIMITATION DES DOMMAGES-INTÉRÊTS ET EXCLUSION DE RESPONSABILITÉ POUR LES DOMMAGES. Vous pouvez obtenir de Microsoft et de ses fournisseurs une indemnisation en cas de dommages directs uniquement à hauteur de 5,00 \$ US. Vous ne pouvez prétendre à aucune indemnisation pour les autres dommages, y compris les dommages spéciaux, indirects ou accessoires et pertes de bénéfices.

Cette limitation concerne:

- tout ce qui est relié au le contenu sous licence, aux services ou au contenu (y compris le code) figurant sur des sites Internet tiers ou dans des programmes tiers; et.
- les réclamations au titre de violation de contrat ou de garantie, ou au titre de responsabilité stricte, de négligence ou d'une autre faute dans la limite autorisée par la loi en vigueur.

Elle s'applique également, même si Microsoft connaissait ou devrait connaître l'éventualité d'un tel dommage. Si votre pays n'autorise pas l'exclusion ou la limitation de responsabilité pour les dommages indirects, accessoires ou de quelque nature que ce soit, il se peut que la limitation ou l'exclusion ci-dessus ne s'appliquera pas à votre égard.

EFFET JURIDIQUE. Le présent contrat décrit certains droits juridiques. Vous pourriez avoir d'autres droits prévus par les lois de votre pays. Le présent contrat ne modifie pas les droits que vous confèrent les lois de votre pays si celles-ci ne le permettent pas.

Revised April 2019



Contents

■	Module 0 About this course	1
	Introduction	1
	About this course	2
	Course agenda	3
	Course audience	9
	Course pre-reqs	10
	Certification details	11
	About the course authors	12
	Module Lab information	15
■	Module 1 Explore compute and storage options for data engineering workloads	17
	Module introduction	17
	Introduction to Azure Synapse Analytics	18
	Introduction to Azure Databricks	26
	Describe Azure Databricks Delta Lake architecture	36
	Introduction to Azure Data Lake storage	44
	Work with data streams by using Azure Stream Analytics (Repo content is updated)	54
	Module Lab information	60
	Module summary	61
■	Module 2 Run interactive queries using serverless SQL pools	67
	Module introduction	67
	Explore Azure Synapse serverless SQL pools capabilities	68
	Query data in the lake using Azure Synapse serverless SQL pools	73
	Create metadata objects in Azure Synapse serverless SQL pools	96
	Secure data and manage users in Azure Synapse serverless SQL pools	105
	Module Lab information	111
	Module summary	112
■	Module 3 Data Exploration and Transformation in Azure Databricks	117
	Module introduction	117
	Describe Azure Databricks	118
	Read and write data in Azure Databricks	129
	Work with DataFrames in Azure Databricks	143
	Work with DataFrames advanced methods in Azure Databricks	156
	Module Lab information	168

Module summary	169
Module 4 Explore, transform, and load data into the Data Warehouse using Apache Spark	173
Module introduction	173
Understand big data engineering with Apache Spark in Azure Synapse Analytics	174
Ingest data with Apache Spark notebooks in Azure Synapse Analytics	184
Transform data with DataFrames in Apache Spark Pools in Azure Synapse Analytics	213
Integrate SQL and Apache Spark pools in Azure Synapse Analytics	230
Monitor manage data engineering workloads with Apache Spark Azure Synapse Analyt	244
Module Lab information	257
Module summary	258
Module 5 Ingest and load Data into the Data Warehouse	263
Module introduction	263
Use data loading best practices in Azure Synapse Analytics	264
Petabyte-scale ingestion with Azure Data Factory	288
Module Lab information	314
Module summary	315
Module 6 Transform Data with Azure Data Factory or Azure Synapse Pipelines	319
Module introduction	319
Data integration with Azure Data Factory	320
Perform code-free transformation at scale with Azure Data Factory	341
Module Lab information	375
Module summary	376
Module 7 Integrate Data from Notebooks with Azure Data Factory or Azure Synapse Pipe-lines	379
Module introduction	379
Orchestrating data movement and transformation in Azure Data Factory	380
Module Lab information	411
Module summary	412
Module 8 End-to-end security with Azure Synapse Analytics	415
Module introduction	415
Secure a data warehouse in Azure Synapse Analytics	416
Configure and manage secrets in Azure Key Vault	448
Implement compliance controls for sensitive data	465
Module Lab information	480
Module summary	481
Module 9 Support Hybrid Transactional Analytical Processing (HTAP) with Azure Synapse Link	485
Module introduction	485
Design hybrid transactional and analytical processing using Azure Synapse Analytics	486
Configure Azure Synapse Link with Azure Cosmos DB	498
Query Azure Cosmos DB with Apache Spark for Azure Synapse Analytics	525
Query Azure Cosmos DB with SQL Serverless for Azure Synapse Analytics	560
Module Lab information	578
Module summary	579
Module 10 Real-time Stream Processing with Stream Analytics	583
Module introduction	583
Enable reliable messaging for Big Data applications using Azure Event Hubs	584
Work with data streams by using Azure Stream Analytics	605

Transform data by using Azure Stream Analytics	611
Module Lab information	629
Module summary	630
Module 11 Create a Stream Processing Solution with Event Hubs and Azure Databricks	633
Module introduction	633
Process streaming data with Azure Databricks structured streaming	634
Module Lab information	660
Module summary	661

Module 0 About this course

Introduction

Course Introduction

This module provides an overview of the following aspects of the course:

- About this course
- Course agenda
- Course audience
- Course pre-requisites
- DP203: Data Engineering on Microsoft Azure certification details
- About the course authors and contributors

About this course

About this course

In this course, the student will learn about the data engineering patterns and practices as it pertains to working with batch and real-time analytical solutions using Azure data platform technologies. Students will begin by understanding the core compute and storage technologies that are used to build an analytical solution. They will then explore how to design an analytical serving layers and focus on data engineering considerations for working with source files.

The students will learn how to interactively explore data stored in files in a data lake. They will learn the various ingestion techniques that can be used to load data using the Apache Spark capability found in Azure Synapse Analytics or Azure Databricks, or how to ingest using Azure Data Factory or Azure Synapse pipelines. The students will also learn the various ways they can transform the data using the same technologies that is used to ingest data.

The student will spend time on the course learning how to monitor and analyze the performance of analytical system so that they can optimize the performance of data loads, or queries that are issued against the systems. They will understand the importance of implementing security to ensure that the data is protected at rest or in transit.

The student will then show how the data in an analytical system can be used to create dashboards or build predictive models in Azure Synapse Analytics.

Course agenda

Course Agenda

At the end of this course, the student will learn:

Module 1: Explore compute and storage options for data engineering workloads

In this module, you will learn how to get familiar with core compute technologies used for data engineering in Azure. You will gain an understanding of the role that a data lake plays in providing a storage layer. You will also learn how to organize your data in a data lake to optimize for exploration, streaming and batch workloads.

Module Objectives:

At the end of this module, the students will be able to:

- Understand Azure Synapse Analytics
- Describe Azure Databricks
- Describe Azure Databricks Delta Lake architecture
- Understand Azure Data Lake storage
- Work with data streams by using Azure Stream Analytics

Module 2: Design and Implement the serving layer

In this module, students will learn how to design and implement a serving layer. You will learn how to create and manage data pipelines in the cloud using Azure Data Factory. You will explore multidimensional schema's to optimize analytical workloads while performing code-free transformations at scale with Azure Data Factory.

Module Objectives:

At the end of this module, the students will be able to:

- Design multidimensional schema's to optimize analytical workloads
- Design a star schema for analytical workloads (OLAP)
- Populate slowly changing dimensions with Azure Data Factory and mapping data flows
- Perform code-free transformation at scale with Azure Data Factory

Module 3: Data engineering considerations for source files

In this module, you will learn how to design a data warehouse using modern architecture patterns, and understand the security approach required to protect it.

Module Objectives:

At the end of this module, the students will be able to:

- Design a Modern Data Warehouse using Azure Synapse Analytics
- Secure a data warehouse in Azure Synapse Analytics

Module 4: Run interactive queries using serverless SQL pools

In this module, you will learn how to query and prepare data in an interactive way on files placed in Azure Storage using Azure Synapse Analytics.

Module Objectives:

At the end of this module, the students will be able to:

- Explore Azure Synapse serverless SQL pools capabilities
- Query data in the lake using Azure Synapse serverless SQL pools
- Create metadata objects in Azure Synapse serverless SQL pools
- Secure data and manage users in Azure Synapse serverless SQL pools

Module 5: Explore, transform, and load data into the Data Warehouse using Apache Spark

In this module, you will learn how to perform data engineering with Azure Synapse Apache Spark pools, which enable you to boost the performance of big-data analytic applications by in-memory cluster computing. You will ingest data with Apache Spark notebooks, transform data with DataFrames, integrate SQL and Apache Spark pools in Azure Synapse Analytics, while monitoring and managing the workloads

Module Objectives:

At the end of this module, the students will be able to:

- Understand big data engineering with Apache Spark pools in Azure Synapse Analytics
- Ingest data with Apache Spark notebooks in Azure Synapse Analytics
- Transform data with DataFrames in Apache Spark pools in Azure Synapse Analytics
- Integrate SQL and Apache Spark pools in Azure Synapse Analytics
- Monitor and manage data engineering workloads with Apache Spark pools in Azure Synapse Analytics

Module 6: Data Exploration and Transformation in Azure Databricks

In this module, students will learn how to harness the power of Apache Spark and powerful clusters running on the Azure Databricks platform to run large data engineering workloads in the cloud combined with Azure Synapse Analytics.

Module Objectives:

At the end of this module, the students will be able to:

- Describe Azure Databricks
- Read and write data in Azure Databricks
- Work with DataFrames in Azure Databricks
- Work with DataFrames advanced methods in Azure Databricks

Module 7: Ingest and load data into the Data Warehouse

In this module, you will learn how to use the best practices you need to adopt to load data into a data warehouse in Azure Synapse Analytics. You will understand the various methods that can be used to ingest data between various data stores using Azure Data Factory or Azure Synapse Pipelines. You will also perform petabyte-scale ingestion with Azure Data Factory or Azure Synapse Pipelines.

Module Objectives:

At the end of this module, the students will be able to:

- Use data loading best practices in Azure Synapse Analytics
- Perform Petabyte-scale ingestion with Azure Data Factory or Azure Synapse Pipelines

Module 8: Transform Data with Azure Data Factory or Azure Synapse Pipelines

In this module, you will learn how to perform data integration with Azure Data Factory or Azure Synapse pipelines. Students will be taught how to execute code-free transformation at scale with Azure Data Factory and Azure Synapse pipelines.

Module Objectives:

At the end of this module, the students will be able to:

- Perform Data integration with Azure Data Factory or Azure Synapse Pipelines
- Perform Code-free transformation at scale with Azure Data Factory or Azure Synapse Pipelines

Module 9: Integrate Data from Notebooks with Azure Data Factory or Azure Synapse Pipelines

In this module, students will learn how to orchestrate data movement and transformations in Azure Data Factory or Azure Synapse Pipeline. Students will learn how Azure Data Factory or Azure Synapse pipelines can orchestrate large scale data movement by using other Azure Data Platform and Machine Learning technologies.

Module Objectives:

At the end of this module, the students will be able to:

- Understand how to add an activity to the control flow to orchestrate data from other technologies

- Understand how to use parameters in Azure Data Factory/Synapse pipelines

Module 10: Optimize Query Performance with Dedicated SQL Pools in Azure Synapse

In this module, students will learn how to optimize query performance with dedicated SQL pools in Azure Synapse Analytics. They will also get familiarized with the techniques that they can use to optimize query performance within Azure Synapse Analytics. Students will also be taught about the language capabilities that are available to create data warehouses in Azure Synapse Analytics and create an understanding of data warehouse developer features of Azure Synapse Analytics.

Module Objectives:

At the end of this module, the students will be able to:

- Optimize data warehouse query performance in Azure Synapse Analytics
- Understand data warehouse developer features of Azure Synapse Analytics

Module 11: Analyze and Optimize Data Warehouse Storage

In this module, you will learn to analyze information used to optimize a data warehouse in Azure Synapse Analytics. You will learn about skewed data, space usage, column storage details, the impact of materialized views and understand logged operations.

Module Objectives:

At the end of this module, the students will be able to:

- Understand skewed data and space usage
- Understand column store storage details
- Understand the impact of wrong choices for column data types
- Describe the impact of materialized views
- Understand rules for minimally logged operations

Module 12: Support Hybrid Transactional Analytical Processing (HTAP) with Azure Synapse Link

In this module, students will learn how to perform operational analytics against Azure Cosmos DB using the Azure Synapse Link feature within Azure Synapse Analytics.

Module Objectives:

At the end of this module, the students will be able to:

- Design hybrid transactional and analytical processing using Azure Synapse Analytics
- Configure Azure Synapse Link with Azure Cosmos DB
- Query Azure Cosmos DB with Apache Spark for Azure Synapse Analytics

- Query Azure Cosmos DB with SQL Serverless for Azure Synapse Analytics

Module 13: End-to-end security with Azure Synapse Analytics

In this module, you will learn how to approach and implement security to protect your data with Azure Synapse Analytics.

Module Objectives:

At the end of this module, the students will be able to:

- Secure a data warehouse
- Configure and manage secrets in Azure Key Vault
- Implement compliance controls for sensitive data

Module 14: Real-time Stream Processing with Stream Analytics

In this module, students will learn the concepts of event processing and streaming data and how this applies to Azure Stream Analytics and Azure Event Hubs. You will understand how to set up a stream analytics job to stream data, and learn how to manage and monitor a running job. You will also learn how to enable reliable messaging for big data applications using Azure Event Hubs.

Module Objectives:

At the end of this module, the students will be able to:

- Work with data streams by using Azure Stream Analytics
- Enable reliable messaging for Big Data applications using Azure Event Hubs
- Ingest data streams with Azure Stream Analytics

Module 15: Create a Stream Processing Solution with Event Hubs and Azure Databricks

This module teaches the student how Structured Streaming helps you process streaming data in real time using Azure Databricks and Azure Event Hubs. This module enables you to aggregate data over windows of time and read and write streams to Azure Event Hubs.

Module Objectives:

At the end of this module, the students will be able to:

- Learn the key features and uses of Structured Streaming.
- Stream data from a file and write it out to a distributed file system.
- Use sliding windows to aggregate over chunks of data rather than all data.
- Apply watermarking to throw away stale old data that you do not have space to keep.

- Connect to Event Hubs read and write streams.

Module 16: Build reports using Power BI integration with Azure Synapse Analytics

In this module, you will learn how you can build Power BI reports from within Azure Synapse Analytics.

Module Objectives:

At the end of this module, the students will be able to:

- Describe the Power BI and Synapse workspace integration
- Configure Power BI optimization options

Module 17: In this module, you will learn how you can build machine learning models from within Azure Synapse Analytics.

In this module, you will learn how you can build machine learning models from within Azure Synapse Analytics.

Module Objectives:

At the end of this module, the students will be able to:

- Describe Azure Machine Learning and how it integrates with an Azure Synapse Analytics workspace
- Describe Azure Machine Learning AutoML
- Understand how to score machine learning models with PREDICT

Course audience

Course Audience

Primary audience

The audience for this course are data engineers, data professionals, data architects, and business intelligence professionals who want to learn about the data platform technologies that exist on Microsoft Azure that can be used to perform data engineering and storage for analytical solutions.#

Secondary audience

The secondary audience for this course are individuals who work with data science or data analyst workload, or who develop applications that deliver content from the data platform technologies that exist on Microsoft Azure

Course pre-reqs

Course Prerequisites

In addition to their professional experience, students who take this training should have technical knowledge equivalent to the following courses:

- **Azure Data Fundamentals¹**

¹ <https://docs.microsoft.com/en-us/learn/certifications/azure-data-fundamentals/>

Certification details

Microsoft Certification

Azure Data Engineer Associate Certification

A candidate for the Azure Data Engineer Associate certification should have subject matter expertise integrating, transforming, and consolidating data from various structured and unstructured data systems into structures that are suitable for building analytics solutions.

Responsibilities for this role include helping stakeholders understand the data through exploration, building and maintaining secure and compliant data processing pipelines by using different tools and techniques. This professional uses various Azure data services and languages to store and produce cleansed and enhanced datasets for analysis.

An Azure Data Engineer also helps ensure that data pipelines and data stores are high-performing, efficient, organized, and reliable, given a specific set of business requirements and constraints. This professional deals with unanticipated issues swiftly and minimizes data loss. An Azure Data Engineer also designs, implements, monitors, and optimizes data platforms to meet the data pipeline needs.

A candidate for this certification must have solid knowledge of data processing languages, such as SQL, Python, or Scala, and they need to understand parallel processing and data architecture patterns.

To gain this certification, you must pass the following exam:

- Exam DP-203: Data Engineering on Microsoft Azure

This course is used to prepare for exam DP 203.

DP-203: Data Engineering on Microsoft Azure

The DP-203 exam is used to test the breadth of skills required to be a data engineer in Microsoft technologies today. Candidates who earn an Azure Data Engineer certification are verified by Microsoft to have the following skills and knowledge for the **DP203 exam²**.

² <https://docs.microsoft.com/en-us/learn/certifications/exams/dp-203>

About the course authors

About the contributors

Given the breadth of expertise required in data engineering, this course has been written by a variety of individuals across the Microsoft Global Technical Learning team, and various Microsoft Product Groups. We have also collaborated with various Microsoft Partners, Microsoft Certified Trainers and Microsoft Most Valuable Professionals to write content that supports real world data engineering scenarios and content that maps to exam DP-203: Data Engineering on Microsoft Azure as listed below. Here we acknowledge the work of our community who have contributed to this content.

Course Authors

Joel Hulen³

General Manager - Training Practice - Solliance⁴

Joel Hulen has more than 26 years of experience in the IT field, gaining expertise in a variety of disciplines ranging from systems engineering to cloud-based computing, and everything in between. He has focused on software development and data engineering for most of his career, developing and architecting software solutions for many platforms, from single-user devices to large-scale enterprise applications and services. His experience draws from supporting and enriching the software profile and data engineering capabilities of financial services companies to the Department of Defense, NASA, as well as a range of businesses from small to large. Joel currently works with Solliance as the Training Practice General Manager and a member of the Data & AI practice.

Charley Hanania⁵

CEO & Principal Data Platform Specialist - QS2 AG - Quality Software Solutions⁶ Microsoft Certified Trainer, Microsoft Most Valuable Professional.

Working on data platforms since the mid 1990's, Charley has been working on all aspects of SQL Server and database platforms in general since version 4 on OS/2, and has supported numerous organisations in database administration, training, development, architecture, cloud infrastructure and operational excellence, and has conducted focused staff mentoring throughout Europe, the USA & Australasia. Charley is a long standing MCT and Microsoft MVP on the Data Platform. He runs a Cloud & Data-focused Services and Application Development Organisation where his team offer Best of Breed services and solutions to various industries around the world.

Daron Yondem⁷

Data Engineer and Solliance consulting partner - Solliance⁸ Microsoft Certified Trainer, Microsoft Regional Director.

Daron is a Microsoft Regional Director and a Microsoft MVP since 2008, with a focus on AI and Azure. He is a regular speaker at international conferences talking about AI, Serverless, and Engineering Culture.

³ <https://www.linkedin.com/in/joel-hulen-ba76b73/>

⁴ <https://solliance.net/>

⁵ <https://mvp.microsoft.com/en-us/PublicProfile/4025576?fullName=Charley%20Hanania>

⁶ <https://www.qs2.ch>

⁷ <https://mvp.microsoft.com/en-us/PublicProfile/4015692?fullName=Daron%20Yondem>

⁸ <https://solliance.net/>

Daron is the founder of Deeload, a Solliance Partner Network Member, and an active Microsoft Certified Trainer (MCT).

Anupama Natarajan⁹

Director/Lead Consultant - Pearl Innovations Limited¹⁰ Microsoft Certified Trainer, Microsoft Most Valuable Professional

Anu is a Data, Analytics and AI Consultant with 20+ years of experience working in design and development of Data Warehouse, Business Intelligence, AI enabled applications and SaaS integrated solutions. She is a Microsoft Certified Trainer (MCT) and a Microsoft MVP in Artificial Intelligence and passionate in sharing knowledge. She enjoys solving complex business problems with innovative solutions using Microsoft technologies and share those experience in her trainings. Anu speaks at conferences (PASS Summit, SQL Saturdays, Data Platform Summit, Difinity), organise local user group meetups and SQLSaturday event organiser in Wellington, New Zealand.

Kyle Bunting¹¹

Data Engineer and Solliance consulting partner - Solliance¹²

Kyle Bunting has been in the IT industry for more than 22 years, with experience ranging from software engineering to cloud-computing, with particular expertise in data and machine learning engineering. He focused on software development for over 18 years, building and architecting enterprise applications and leading development teams across multiple industries and continents, including the Department of Defense, insurance, and digital media. Since joining Solliance, his primary focus is on helping organizations modernize their data architectures and applications and take advantage of the many benefits of migrating to the cloud. He received his M.B.A. from Purdue University's Krannert School of Management in 2016 and has a B.A. in Psychology from Purdue University. He also has an A.A.S. in Computer Programming and Network Engineering from ECPI University.

Dustin Vannoy¹³

Data Engineer and Solliance consulting partner - Solliance¹⁴

Dustin Vannoy is a Data Engineer and Solliance consulting partner. He has 15 years experience solving business problems with analytics and big data solutions. He is passionate about all aspects of data engineering, especially building data platforms and streaming data pipelines. He currently focuses on building data lakes and data pipelines in Azure with Apache Spark, Python, and Scala. He is co-founder of the Data Engineering San Diego meetup and participates in other community learning groups.

Technical Reviewer

Mark Fitzgerald¹⁵

Principal Technical Learning Specialist - QA¹⁶ Microsoft Certified Trainer

⁹ <https://mvp.microsoft.com/en-us/PublicProfile/5003080?fullName=Anupama%20Natarajan>

¹⁰ <https://pearlinnovations.co.nz/>

¹¹ <https://www.linkedin.com/in/kylebunting/>

¹² <https://solliance.net/>

¹³ <https://www.linkedin.com/in/dustinvannoy/>

¹⁴ <https://solliance.net/>

¹⁵ <https://www.linkedin.com/in/mark-fitzgerald-58aa3919/?originalSubdomain=uk>

¹⁶ www.qa.com

Mark is a Principal Technical Learning Specialist within the Microsoft SQL and Azure Data working for QA Ltd in the UK. Mark joined QA Ltd in 2000 and has worked in the data and analytics arena for 35 years. He has worked in development, administration, support or advisory roles in various sectors, including food production, automotive manufacture and energy services. He covers all the Microsoft SQL Server curriculum from SQL 7.0 to SQL 2019, Power BI, and has been leading the Azure data platform curriculum within QA. Mark has authored twelve courses that are delivered by QA. He loves challenges, working with data to provide the best results for the customer.

Course advisors

In creating this course, we also sought counsel from the following Microsoft Certified Trainers:

- **Glenn Morris¹⁷** Microsoft Certified Trainer
- **Dwight Goins¹⁸** Microsoft Certified Trainer/Microsoft Regional Director
- **Michael Johnson¹⁹** Microsoft Certified Trainer/Microsoft Most Valuable Professional

¹⁷ <https://www.linkedin.com/in/glennm/?originalSubdomain=au>

¹⁸ <https://rd.microsoft.com/en-us/dwight-goins>

¹⁹ <https://mvp.microsoft.com/en-us/PublicProfile/5002619>

Module Lab information

Lab 0 - DP-203 course lab setup

Estimated Time: 50 - 60 minutes

The instructions for this lab are located in the Instructions\Labs\00 folder.

Overview

The instructions enables learners to prepare their lab environments for the modules that follow. Please run through these instructions prior to starting Module 1.

Time to complete: It takes around 5 minutes to perform the steps below and initiate the automated setup scripts. The scripts may take an hour or more to complete.

Module 1 Explore compute and storage options for data engineering workloads

Module introduction

module-introduction

In this module, you will learn how to get familiar with core compute technologies used for data engineering in Azure. You will gain an understanding of the role that a data lake plays in providing a storage layer. You will also learn how to organize your data in a data lake to optimize for exploration, streaming and batch workloads.

Learning objectives

In this module, you will:

- Understand Azure Synapse Analytics
- Describe Azure Databricks
- Describe Azure Databricks Delta Lake architecture
- Understand Azure Data Lake storage
- Work with data streams by using Azure Stream Analytics

Introduction to Azure Synapse Analytics

Introduction

Learn how Azure Synapse Analytics solves the issue of having a single service to fulfill the broad range of analytics requirements that organizations face today.

In this lesson, you will learn:

- What is Azure Synapse Analytics
- How Azure Synapse Analytics work
- When to use Azure Synapse Analytics

Before taking this lesson, it is recommended that the student is able to:

- Log into the Azure portal
- Explain and create resource groups

What is Azure Synapse Analytics

Azure Synapse Analytics is an integrated analytics platform, which combines data warehousing, big data analytics, data integration, and visualization into a single environment. Azure Synapse Analytics empowers users of all abilities to gain access and quick insights across all of their data, enabling a whole new level of performance and scale.

Gartner defines a range of analytical types that Azure Synapse Analytics can support including:

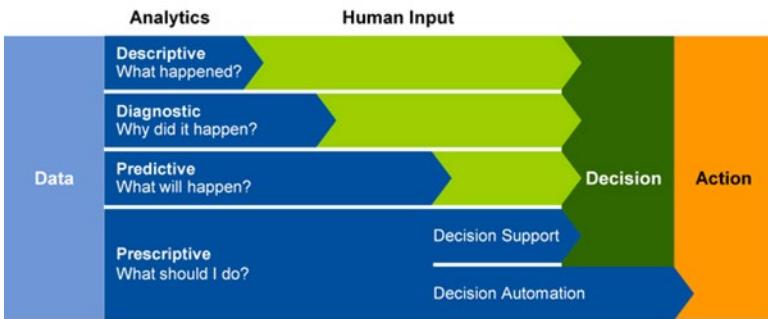
Descriptive analytics

Descriptive analytics answers the question "What is happening in my business?" The data to answer this question is typically answered through the creation of a data warehouse. Azure Synapse Analytics leverages the dedicated SQL pool capability that enables you to create a persisted data warehouse to perform this type of analysis. You can also make use of the serverless SQL pool to prepare data from files stored in a data lake to create a data warehouse interactively.

Diagnostic analytics

Diagnostic analytics deals with answering the question "Why is it happening?". This may involve exploring information that already exists in a data warehouse, but typically involves a wider search of your data estate to find more data to support this type of analysis.

You can use the same serverless SQL pool capability within Azure Synapse Analytics that enables you to interactively explore data within a data lake. Serverless SQL pools can quickly enable a user to search for additional data that may help them to understand "Why is it happening?"



Predictive analytics

Azure Synapse Analytics also enables you to answer the question "What is likely to happen in the future based on previous trends and patterns?" by using its integrated Apache Spark engine. Azure Synapse Spark pools can be used with other services such as Azure Machine Learning Services, or Azure Databricks.

Prescriptive analytics

This type of analytics looks at executing actions based on real-time or near real-time analysis of data, using predictive analytics. Azure Synapse Analytics provides this capability through both Apache Spark, Azure Synapse Link, and by integrating streaming technologies such as Azure Stream Analytics.

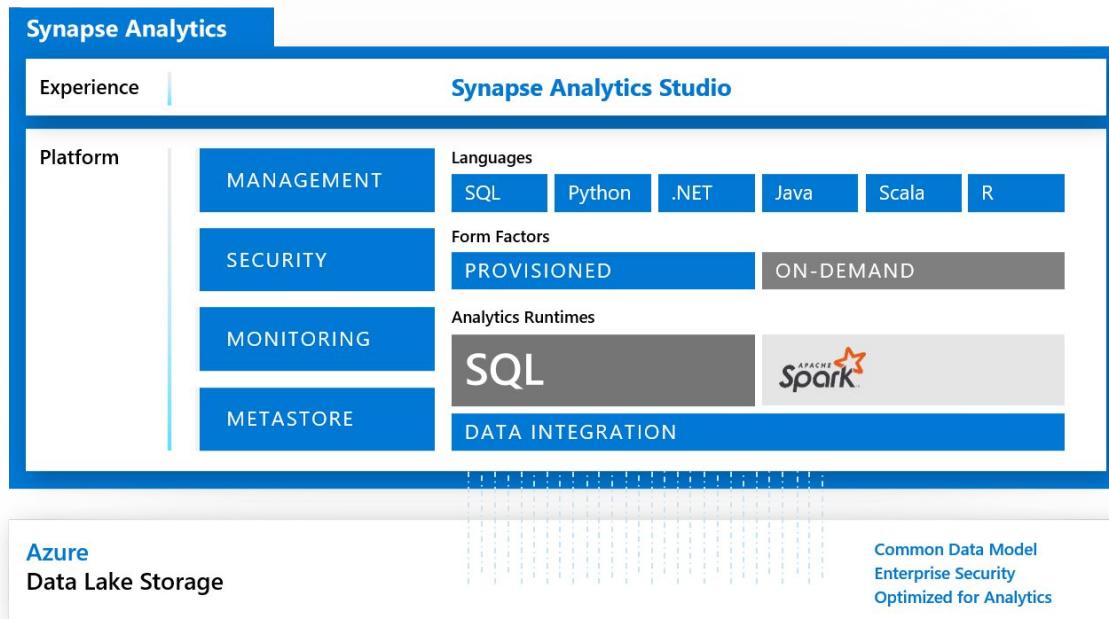
Azure Synapse Analytics gives the users of the service the freedom to query data on their own terms, using either serverless or dedicated resources at scale. Azure Synapse Analytics brings these two worlds together with a unified data integration experience to ingest, prepare, manage, and serve data using Azure Synapse Pipelines. In addition, you can visualize the data in the form of dashboards and reports for immediate analysis using Power BI, which is integrated into the service too.

How Azure Synapse Analytics works

Azure Synapse Analytics can work by acting as the one stop shop to meet all of your analytical needs in an integrated environment if you do not have an analytical environment in place already. It does this by providing the following capabilities:

Analytics capabilities offered through Azure Synapse SQL through either dedicated SQL pools or serverless SQL pools

Azure Synapse SQL is a distributed query system that enables you to implement data warehousing and data virtualization scenarios using standard T-SQL experiences familiar to data engineers. Synapse SQL offers both serverless and dedicated resource models to work with both descriptive and diagnostic analytical scenarios. For predictable performance and cost, create dedicated SQL pools to reserve processing power for data stored in SQL tables. For unplanned or ad-hoc workloads, use the always-available, serverless SQL endpoint.



Apache Spark pool with full support for Scala, Python, SparkSQL, and C#

You can develop big data engineering and machine learning solutions using Apache Spark for Azure Synapse. You can take advantage of the big data computation engine to deal with complex compute transformations that would take too long in a data warehouse.

For machine learning workloads, you can use SparkML algorithms and AzureML integration for Apache Spark 2.4 with built-in support for Linux Foundation Delta Lake.

There is a simple model for provisioning and scaling the Spark clusters to meet your compute needs, regardless of the operations that you are performing on the data.

Data integration with Azure Synapse Pipelines

Azure Synapse Pipelines leverages the capabilities of Azure Data Factory and is the cloud-based ETL and data integration service that allows you to create data-driven workflows for orchestrating data movement and transforming data at scale. Using Azure Synapse Pipelines, you can create and schedule data-driven workflows (called pipelines) that can ingest data from disparate data stores. You can build complex ETL processes that transform data visually with data flows or by using compute services such as Azure Databricks.

Perform operational analytics with near real-time hybrid transactional and analytical processing with Azure Synapse Link

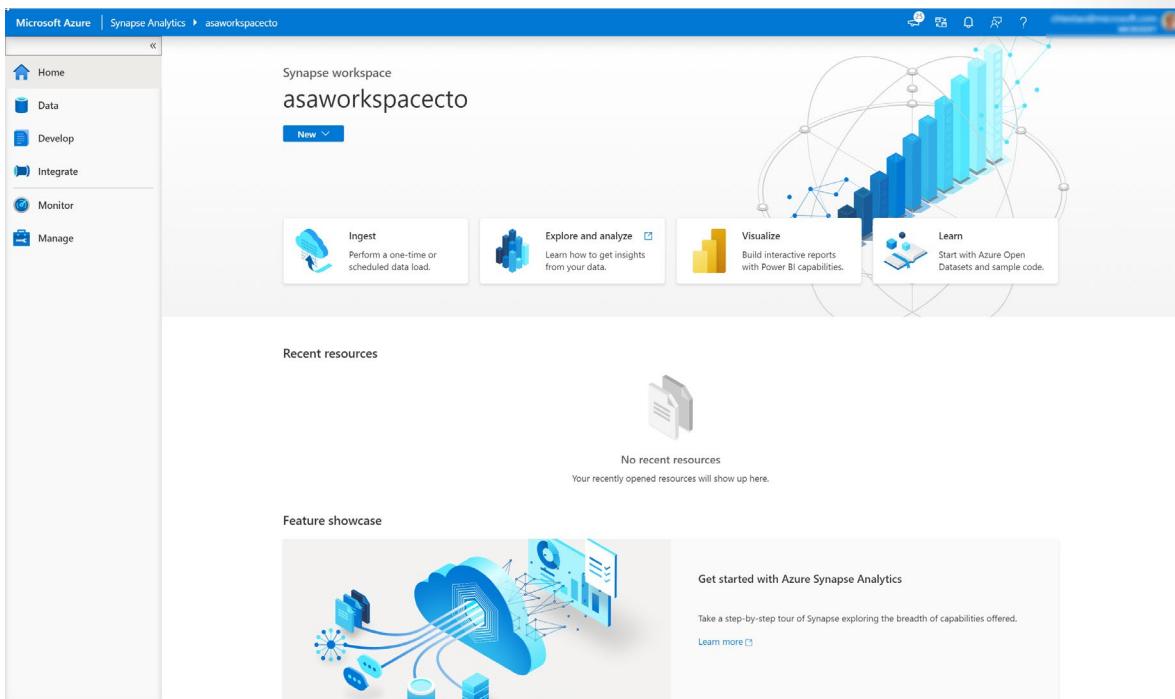
Azure Synapse Analytics enables you to reach out to operational data using Azure Synapse Link, and is achieved without impacting the performance of the transactional data store. For this to happen, you have to enable the feature within both Azure Synapse Analytics, and within the data store to which Azure

Synapse Analytics will connect, such as Azure Cosmos DB.

In the case of Azure Cosmos DB, this will create an analytical data store. As data changes in the transactional system, the changed data is fed to the analytical store in a Column store format from which Azure Synapse Link can query with no disruption to the source system.

A single Web UI to be able to access all Azure Synapse Analytics capabilities

While the Azure portal will allow you to manage some aspects of the service, Azure Synapse Studio is the best place to centrally work with all the capabilities.



Azure Synapse Studio is a single web UI that allows you to:

- Explore your data estate.
- Develop TSQL scripts and notebooks to interact with the analytical engines.
- Build data integration pipelines for managing data movement.
- Monitor the workloads within the service.
- Manage the components of the service.

Integrate with a variety of Azure Data Platform technologies

For organizations that have existing analytical solutions, Azure Synapse Analytics can integrate with a wide variety of technologies to complement them.

For example, if you are already using Azure Data Factory to build data integration pipelines, these can be used to load data into Azure Synapse Analytics. You can also integrate existing data preparation or data science projects that you may hold in Azure Databricks. There is also integration with many Azure security components to ensure that you meet security and compliance requirements within your organization.

On the initial deployment of Azure Synapse Analytics, there are a few resources that deploy along with it, including the Azure Synapse Workspace and an Azure Data Lake Storage Gen2 (ADLS Gen2) account that acts as the primary storage for the workspace.

The screenshot shows the Azure Synapse Analytics workspace settings page. It includes sections for Overview, Activity log, Access control (IAM), Tags, Settings (SQL Active Directory admin, Properties, Locks), Analytics pools (SQL pools, Apache Spark pools), Security (Firewalls, Managed identities), Monitoring (Alerts, Metrics), Automation (Tasks), Support + troubleshooting, and New support request. Key highlighted areas include:

- Primary ADLS Gen2 account URL (1):** https://asadatalakeceto.dfs.core.windows.net
- Dedicated SQL endpoint and Serverless SQL endpoint (2):** asaworkspacetoso.sql.azuresynapse.net and asaworkspacetoso-on-demand.sql.azuresynapse.net
- Workspace web URL (3):** https://web.azuresynapse.net/workspace=%2fsubscriptions%2f...
- Available resources (4):** A table showing SQL pools (Built-in, SQLPool01) and Apache Spark pools (SparkPool01).

Within the Azure portal, there are links to configure your workspace, manage access through Access control (IAM), firewalls, managed identities, and private endpoint connections.

It also contains important information about your Synapse Analytics environment, such as:

- The **Primary ADLS Gen2 account URL (1)**, which identifies the primary data lake storage account.
- The **SQL endpoint** and **SQL on-demand endpoint (2)**, which are used to integrate with external tools, such as SQL Server Management Studio (SSMS), Azure Data Studio, and Power BI.
- The **Workspace web URL (3)**, a direct link to Synapse Studio for the workspace.
- Available resources, such as **SQL pools** and **Apache Spark pools (4)**.

When to use Azure Synapse Analytics

Across all organizations and industries, the common use cases for Azure Synapse Analytics are identified by the need for:

Modern data warehousing

This involves the ability to integrate all data, including big data, to reason over data for analytics and reporting purposes from a descriptive analytics perspective, independent of its location or structure.

Advanced analytics

Enables organizations to perform predictive analytics using both the native features of Azure Synapse Analytics, and integrating with other technologies such as Azure Databricks.

Data exploration and discovery

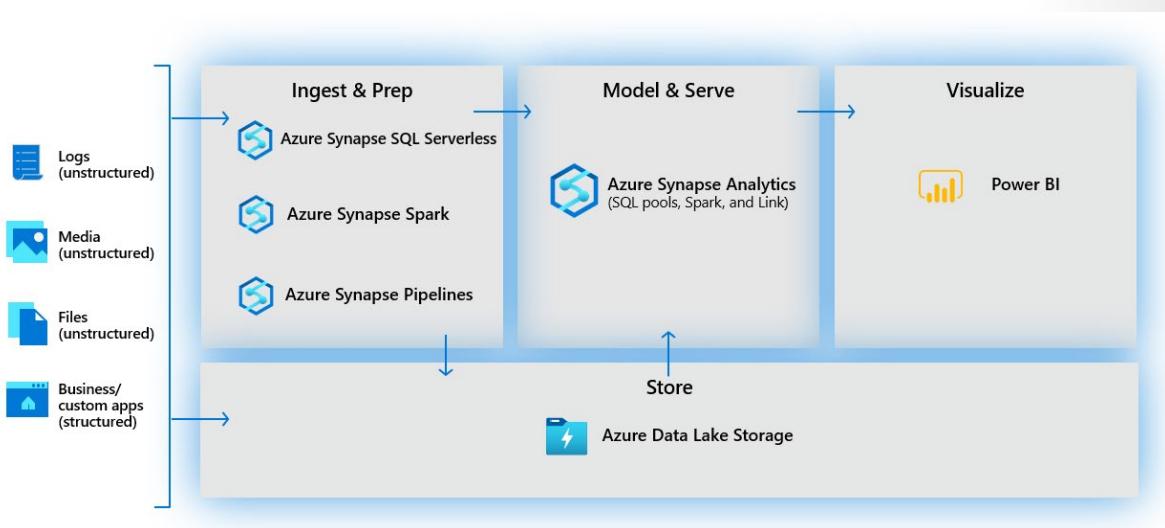
The serverless SQL pool functionality provided by Azure Synapse Analytics enables Data Analysts, Data Engineers and Data Scientist alike to explore the data within your data estate. This capability supports data discovery, diagnostic analytics, and exploratory data analysis.

Real time analytics

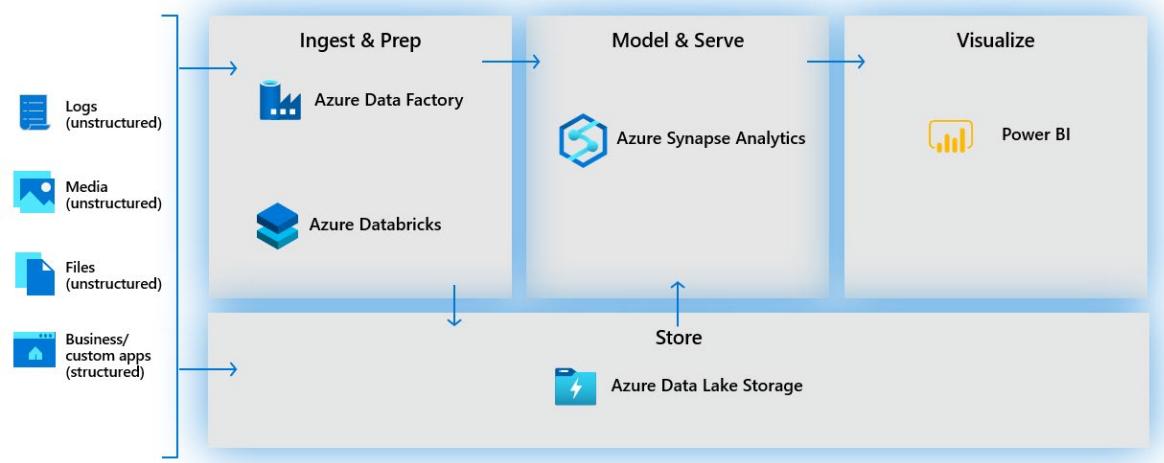
Azure Synapse Analytics can capture, store and analyze data in real-time or near-real time with features such as Azure Synapse Link, or through the integration of services such as Azure Stream Analytics and Azure Data Explorer.

Data integration

Azure Synapse Pipelines enables you to ingest, prepare, model and serve the data to be used by downstream systems. This can be used by components of Azure Synapse Analytics exclusively.



It can also interact with existing Azure services that you may already have in place for your existing analytical solutions.



Integrated analytics

With the variety of analytics that can be performed on the data at your disposal, putting together the services in a cohesive solution can be a complex operation. Azure Synapse Analytics removes this complexity by integrating the analytics landscape into one service. That way you can spend more time working with the data to bring business benefit, than spending much of your time provisioning and maintaining multiple systems to achieve the same outcomes.

Knowledge check

Question 1

Which type of analytics answers the question "What is likely to happen in the future based on previous trends and patterns?

- Descriptive.
- Diagnostic.
- Predictive.

Question 2

You have a requirement to occasionally prepare data for ad hoc data exploration and analysis. Which resource model in Azure Synapse Analytics is the most effective to use to meet this requirement?

- Serverless.
- Dedicated.
- Pipelines.

Question 3

Where can you develop TSQL scripts and notebooks in Azure Synapse Analytics?

- Azure portal.
- Azure Synapse Studio.
- Data Lake.

Summary

In this lesson, you have explored how Azure Synapse Analytics solves the issue of having a single service to fulfill the broad range of analytics requirements that organizations face today.

In this lesson, you have learned:

- What is Azure Synapse Analytics
- How Azure Synapse Analytics work
- When to use Azure Synapse Analytics

Introduction to Azure Databricks

Introduction

Azure Databricks is a fully-managed version of the open-source **Apache Spark¹** analytics and data processing engine. Azure Databricks is an enterprise-grade and secure cloud-based big data and machine learning platform.

Databricks provides a notebook-oriented Apache Spark as-a-service workspace environment, making it easy to manage clusters and explore data interactively.

Learning objectives

In this lesson, you will:

- Understand the Azure Databricks platform
- Create your own Azure Databricks workspace
- Create a notebook inside your home folder in Databricks
- Understand the fundamentals of Apache Spark notebook
- Create, or attach to, a Spark cluster
- Identify the types of tasks well-suited to the unified analytics engine Apache Spark

Explain Azure Databricks

Azure Databricks is a fully-managed, cloud-based Big Data and Machine Learning platform, which empowers developers to accelerate AI and innovation by simplifying the process of building enterprise-grade production data applications. Built as a joint effort by the team that started Apache Spark and Microsoft, Azure Databricks provides data science and engineering teams with a single platform for Big Data processing and Machine Learning.

By combining the power of Databricks, an end-to-end, managed Apache Spark platform optimized for the cloud, with the enterprise scale and security of Microsoft's Azure platform, Azure Databricks makes it simple to run large-scale Spark workloads.

Optimized environment

To address the problems seen on other Big Data platforms, Azure Databricks was optimized from the ground up, with a focus on performance and cost-efficiency in the cloud. The Databricks Runtime adds several key capabilities to Apache Spark workloads that can increase performance and reduce costs by as much as 10-100x when running on Azure, including:

- High-speed connectors to Azure storage services, such as Azure Blob Store and Azure Data Lake
- Auto-scaling and auto-termination of Spark clusters to minimize costs
- Caching
- Indexing
- Advanced query optimization

¹ <https://spark.apache.org/>

By providing an optimized, easy to provision and configure environment, Azure Databricks gives developers a performant, cost-effective platform that enables them to spend more time building applications, and less time focused on managing clusters and infrastructure.

Who is Databricks?

Databricks was founded by the creators of Apache Spark, Delta Lake, and MLflow.

Over 2000 global companies use the Databricks platform across big data & machine learning lifecycle.

Databricks Vision: Accelerate innovation by unifying data science, data engineering and business.

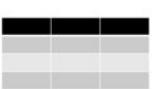
Databricks Solution: Big Data Analytics Platform

What does Databricks offer that is not Open-Source Spark?

- Databricks Workspace - Interactive Data Science & Collaboration
- Databricks Workflows - Production Jobs & Workflow Automation
- Databricks Runtime
- Databricks I/O (DBIO) - Optimized Data Access Layer
- Databricks Serverless - Fully Managed Auto-Tuning Platform
- Databricks Enterprise Security (DBES) - End-To-End Security & Compliance

What is Apache Spark?

Spark is a unified processing engine that can analyze big data using SQL, machine learning, graph processing, or real-time stream analysis:



SQL / DataFrames



Machine Learning



Graph



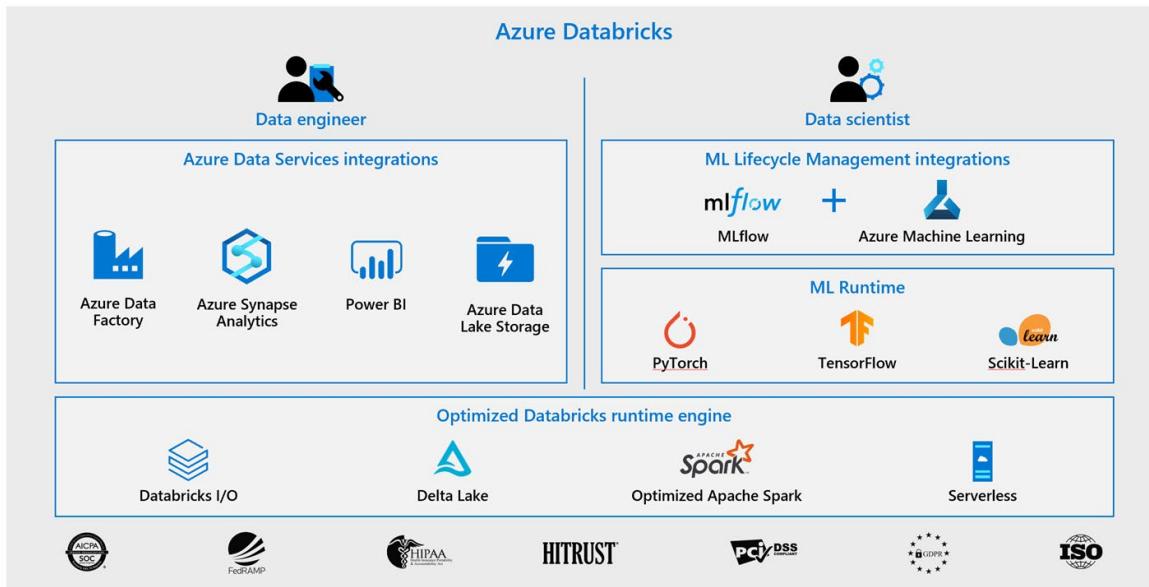
Streaming

- At its core is the Spark Engine.
- The DataFrames API provides an abstraction above RDDs while simultaneously improving performance 5-20x over traditional RDDs with its Catalyst Optimizer.
- Spark ML provides high quality and finely tuned machine learning algorithms for processing big data.
- The Graph processing API gives us an easily approachable API for modeling pairwise relationships between people, objects, or nodes in a network.

- The Streaming APIs give us End-to-End Fault Tolerance, with Exactly-Once semantics, and the possibility for sub-millisecond latency.

And it all works together seamlessly!

Azure Databricks



As a compute engine, Azure Databricks sits at the center of your Azure-based software platform and provides native integration with Azure Active Directory (Azure AD) and other Azure services.

Scala, Python, Java, R & SQL

- Besides being able to run in many environments, Apache Spark makes the platform even more approachable by supporting multiple languages:
 - Scala - Apache Spark's primary language
 - Python - More commonly referred to as PySpark
 - R - **SparkR²** (R on Spark)
 - Java
 - SQL - Closer to ANSI SQL 2003 compliance
 - Now running all 99 TPC-DS queries
 - New standards-compliant parser (with good error messages!)
 - Subqueries (correlated & uncorrelated)
 - Approximate aggregate stats

² <https://spark.apache.org/docs/latest/sparkr.html>

- With the DataFrames API, the performance differences between languages are nearly nonexistence (especially for Scala, Java & Python).

Create an Azure Databricks workspace and cluster

When talking about the Azure Databricks workspace, we refer to two different things. The first reference is the logical Azure Databricks environment in which clusters are created, data is stored (via DBFS), and in which the server resources are housed. The second reference is the more common one used within the context of Azure Databricks. That is the special root folder for all of your organization's Databricks assets, including notebooks, libraries, and dashboards, as shown below:

The screenshot shows the Microsoft Azure Databricks workspace interface. On the left is a dark sidebar with icons for Azure Databricks, Home, Workspace, Recents, Data, Clusters, Jobs, and Search. The main area has a title 'Workspace' and dropdown menus for 'Workspace' (set to 'Shared') and 'Users'. The 'Users' dropdown is expanded, showing a list of users: joel@ (selected), databricks01@, databricks02@, databricks03@, databricks04@, databricks05@, databricks06@, databricks07@, databricks08@, databricks09@, databricks10@, databricks11@, databricks12@, databricks13@, databricks14@, databricks15@, and databricks16@. Each user entry has a three-dot menu icon to its right.

The first step to using Azure Databricks is to create and deploy a Databricks workspace, which is the logical environment. You can do this in the Azure portal.

Deploy an Azure Databricks workspace

1. Open the Azure portal.
2. Click **Create a Resource** in the top left
3. Search for "Databricks"
4. Select *Azure Databricks*
5. On the Azure Databricks page select *Create*
6. Provide the required values to create your Azure Databricks workspace:
 - **Subscription:** Choose the Azure subscription in which to deploy the workspace.
 - **Resource Group:** Use **Create new** and provide a name for the new resource group.
 - **Location:** Select a location near you for deployment. For the list of regions that are supported by Azure Databricks, see **Azure services available by region³**.
 - **Workspace Name:** Provide a unique name for your workspace.
 - **Pricing Tier: Trial (Premium - 14 days Free DBUs)**. You must select this option when creating your workspace or you will be charged. The workspace will suspend automatically after 14 days. When the trial is over you can convert the workspace to **Premium** but then you will be charged for your usage.
7. Select **Review + Create**.
8. Select **Create**.

The workspace creation takes a few minutes. During workspace creation, the **Submitting deployment for Azure Databricks** tile appears on the right side of the portal. You might need to scroll right on your dashboard to see the tile. There's also a progress bar displayed near the top of the screen. You can watch either area for progress.

What is a cluster?

The notebooks are backed by clusters, or networked computers, that work together to process your data. The first step is to create a cluster.

Create a cluster

1. When your Azure Databricks workspace creation is complete, select the link to go to the resource.
2. Select **Launch Workspace** to open your Databricks workspace in a new tab.
3. In the left-hand menu of your Databricks workspace, select **Clusters**.
4. Select **Create Cluster** to add a new cluster.

³ <https://azure.microsoft.com/regions/services/>

Create Cluster

New Cluster

[Cancel](#)

[Create Cluster](#)

2-8 Workers: 28.0-112.0 GB Memory, 8-32 Cores
1 Driver: 14.0 GB Memory, 4 Cores, 0.75 DBU

Cluster Name

Test Cluster



Cluster Mode

Standard

Pool

None

Databricks Runtime Version

[Learn more](#)

Runtime: 6.4 (Scala 2.11, Spark 2.4.5)

New This Runtime version supports only Python 3.

Autopilot Options

Enable autoscaling

Terminate after minutes of inactivity

Worker Type

Standard_DS3_v2

14.0 GB Memory, 4 Cores, 0.75 DBU

Min Workers

2

Max Workers

8

Driver Type

Same as worker

14.0 GB Memory, 4 Cores, 0.75 DBU

- Enter a name for your cluster. Use your name or initials to easily differentiate your cluster from your coworkers.
- Select the **Databricks RuntimeVersion**. We recommend the latest runtime and **Scala 2.11**.
- Specify your cluster configuration. While on the 14 day free trial, the defaults will be sufficient. When the trial is ended, you may prefer to change **Min Workers** to zero. That will allow the compute resources to shut down when you are not in a coding exercise and reduce your charges.

Hint: Check with your local system administrator to see if there is a recommended default cluster at your company to use for the rest of the class. This could save you some money!

- Select **Create Cluster**.

Understand Azure Databricks Notebooks

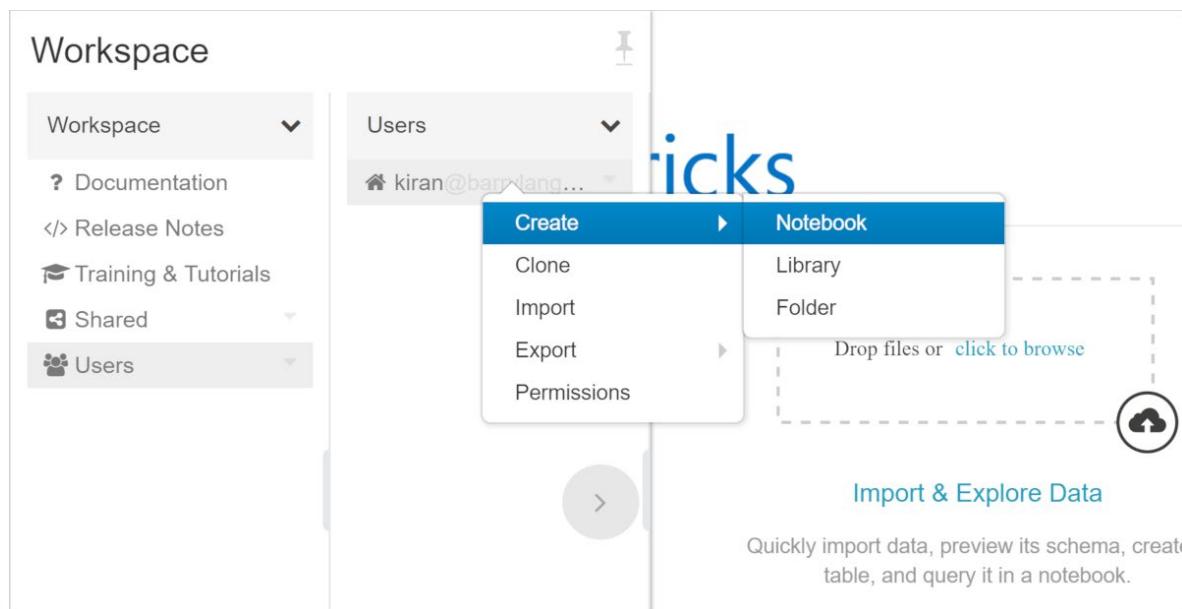
After creating your Databricks workspace, it's time to create your first notebook. To execute your notebook, you will attach the cluster you created in the previous unit.

What is Apache Spark notebook?

A notebook is a collection of cells. These cells are run to execute code, to render formatted text, or to display graphical visualizations.

Create a notebook

1. In the Azure portal, click **All resources** menu on the left side navigation and select the Databricks workspace you created in the last unit.
2. Select **Launch Workspace** to open your Databricks workspace in a new tab.
3. On the left-hand menu of your Databricks workspace, select **Home**.
4. Right-click on your home folder.
5. Select **Create**.
6. Select **Notebook**.



7. Name your notebook **First Notebook**.
8. Set the **Language** to **Python**.
9. Select the cluster to which to attach this notebook.

NOTE:

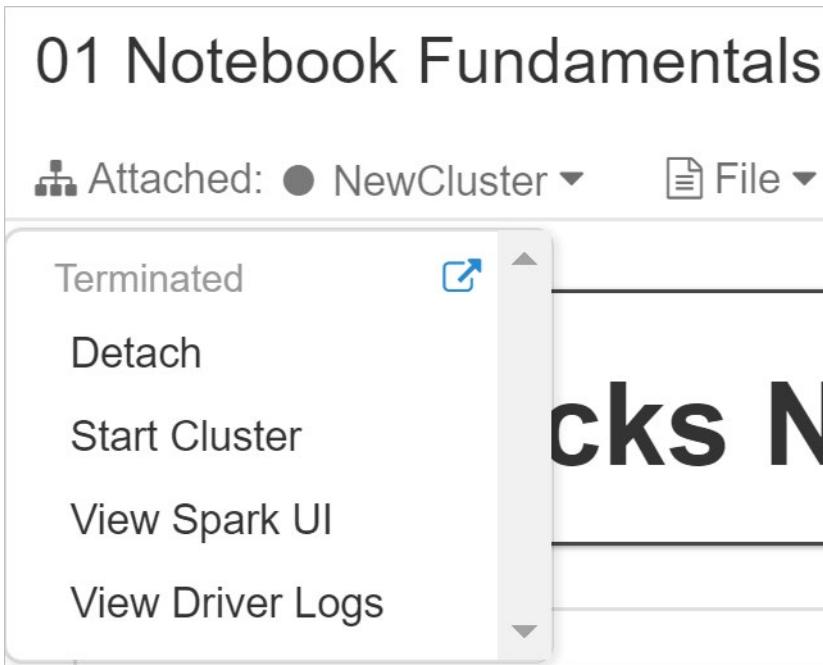
This option displays only when a cluster is currently running. You can still create your notebook and attach it to a cluster later.

10. Select **Create**.

Now that you've created your notebook, let's use it to run some code.

Attach and detach your notebook

To use your notebook to run a code, you must attach it to a cluster. You can also detach your notebook from a cluster and attach it to another depending upon your organization's requirements.



If your notebook is attached to a cluster, you can:

- Detach your notebook from the cluster
- Restart the cluster
- Attach to another cluster
- Open the Spark UI
- View the log files of the driver

Exercise - Work with Notebooks

You can use Apache Spark notebooks to:

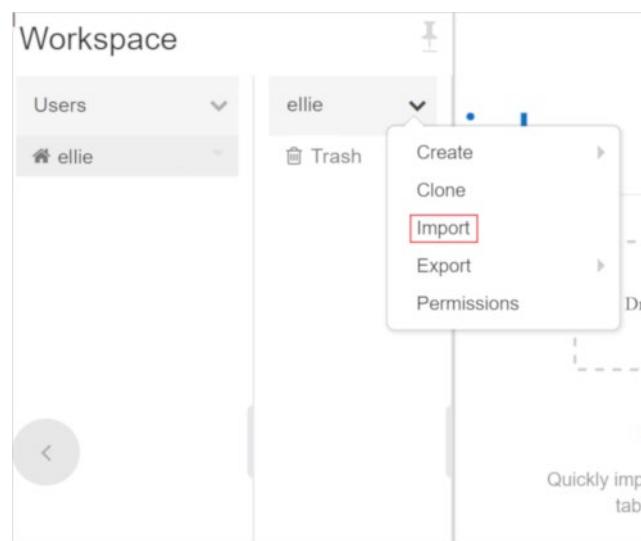
- Read and process huge files and data sets
- Query, explore, and visualize data sets
- Join disparate data sets found in data lakes
- Train and evaluate machine learning models
- Process live streams of data
- Perform analysis on large graph data sets and social networks

To learn more about using notebooks, clone the labs archive where sample notebooks are provided. These notebooks will help you understand how to use notebooks for your day-to-day tasks.

Clone the Databricks archive

1. In the Azure portal, navigate to your deployed Azure Databricks workspace and select **Launch Workspace**.
2. In the left pane, select **Workspace > Users**, and select your username (the entry with the house icon).

3. In the pane that appears, select the arrow next to your name, and select **Import**.



4. In the **Import Notebooks** dialog box, select the URL and paste in the following URL:

<https://github.com/solliancenet/microsoft-learning-paths-databricks-notebooks/blob/master/data-engineering/DBC/01-Introduction-to-Azure-Databricksdbc?raw=true>

1. Select **Import**.
2. Select the **01-Introduction-to-Azure-Databricks** folder that appears.
3. Use the set of notebooks in this folder to complete this lab.

Complete the following notebook

- **01-The-Databricks-Environment** - This notebook illustrates the fundamentals of a Databricks notebook.

Knowledge check

Question 1

How many drivers does a Cluster have?

- Only one
- Two, running in parallel
- Configurable between one and eight

Question 2

Spark is a distributed computing environment. Therefore, work is parallelized across executors. At which two levels does this parallelization occur?

- The Executor and the Slot
- The Driver and the Executor
- The Slot and the Task

Question 3

What type of process are the driver and the executors?

- Java processes
- Python processes
- C++ processes

Summary

In this module, you learned the basics about using Databricks workspace and Apache Spark notebooks. The notebooks allow you to interactively work with different types of data. You can use notebooks to process huge data files, query, read and write data from different sources, train machine learning models, and process live data streams.

Now that you have concluded this module, you should know:

1. How to Create a Cluster
2. How to Attach a Notebook to a Cluster
3. How to Execute Python, Scala, SQL or R code in a Notebook Cell
4. How to Detach a Notebook from a Cluster
5. How Azure Databricks fits into the Azure Ecosystem
6. How DBFS works to present Blob Storage as a Filesystem

Clean up

If you plan on completing other Azure Databricks modules, don't delete your Azure Databricks instance yet. You can use the same environment for the other modules.

Delete the Azure Databricks instance

1. Navigate to the Azure portal.
2. Navigate to the resource group that contains your Azure Databricks instance.
3. Select **Delete resource group**.
4. Type the name of the resource group in the confirmation text box.
5. Select **Delete**.

Describe Azure Databricks Delta Lake architecture

Introduction

Imagine that you work in the IT department of a large retail store. Your organization uses Azure Data Lake to store all its online shopping data. However, as the volume of data increases, updating and querying information is becoming more and more time consuming. Your responsibility is to investigate the problem and find a solution.

In addition, your batch processing needs to be augmented with stream processing to gain insights on user clickstream data from online shoppers. As a result, your solution also needs to handle the emerging stream processing requirements. You have investigated using a Lambda architecture, which is a big data processing architecture that combines both batch- and real-time processing methods. However, it is difficult to combine processing of batch and real-time data using a traditional Lambda architecture.

You need a solution that matches Data Lake in scalability but is also reliable and fast. You also need your solution that effectively combines batch and real-time data processing.

Delta Lake can solve your problem. It's a unified data-management system that combines the best capabilities of Data Lake, data warehousing, and a streaming ingestion system.

Learning objectives

In this module, you will:

- Process batch and streaming data with Delta Lake.
- Learn how Delta Lake architecture enables unified streaming and batch analytics with transactional guarantees within a data lake.

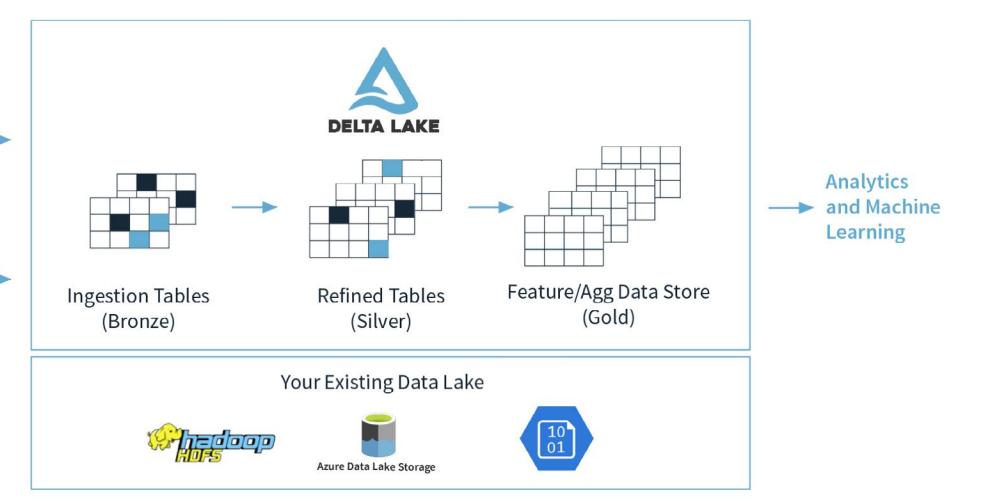
Prerequisites

None

Describe bronze, silver, and gold architecture

This module introduces using Delta Lakes as an optimization layer on top of blob storage to ensure reliability, ACID compliance and low latency within unified Streaming + Batch data pipelines.

We will discuss this relative to a traditional Lambda architecture.



An example of a Delta Lake Architecture might be as shown in the diagram above.

- Many **devices** generate data across different ingestion paths.
- Streaming data can be ingested from **IOT Hub** or **Event Hub**.
- Batch data can be ingested by **Azure Data Factory** or **Azure Databricks**.
- Extracted, Transformed data is loaded into a **Delta Lake**.

Lambda architecture

When working with large data sets, it can take a long time to run the sort of queries that clients need. These queries can't be performed in real time, and often require algorithms such as **MapReduce**⁴ that operate in parallel across the entire data set. The results are then stored separately from the raw data and used for querying.

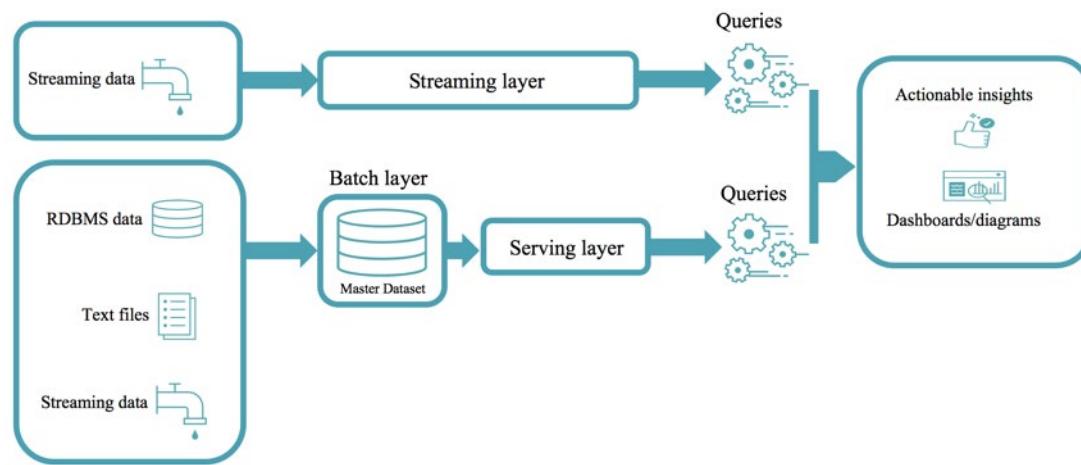
One drawback to this approach is that it introduces latency. If processing takes a few hours, a query may return results that are several hours old. Ideally, you would like to get some results in real time (perhaps with some loss of accuracy), and combine these results with the results from the batch analytics.

The **lambda architecture** is a big data processing architecture that addresses this problem by combining both batch- and real-time processing methods. It features an append-only immutable data source that serves as system of record. Timestamped events are appended to existing events (nothing is overwritten). Data is implicitly ordered by time of arrival.

Notice how there are really two pipelines here, one batch and one streaming, hence the name *lambda* architecture.

It is difficult to combine processing of batch and real-time data as is evidenced by the diagram below:

⁴ <https://en.wikipedia.org/wiki/MapReduce>



Delta Lake architecture

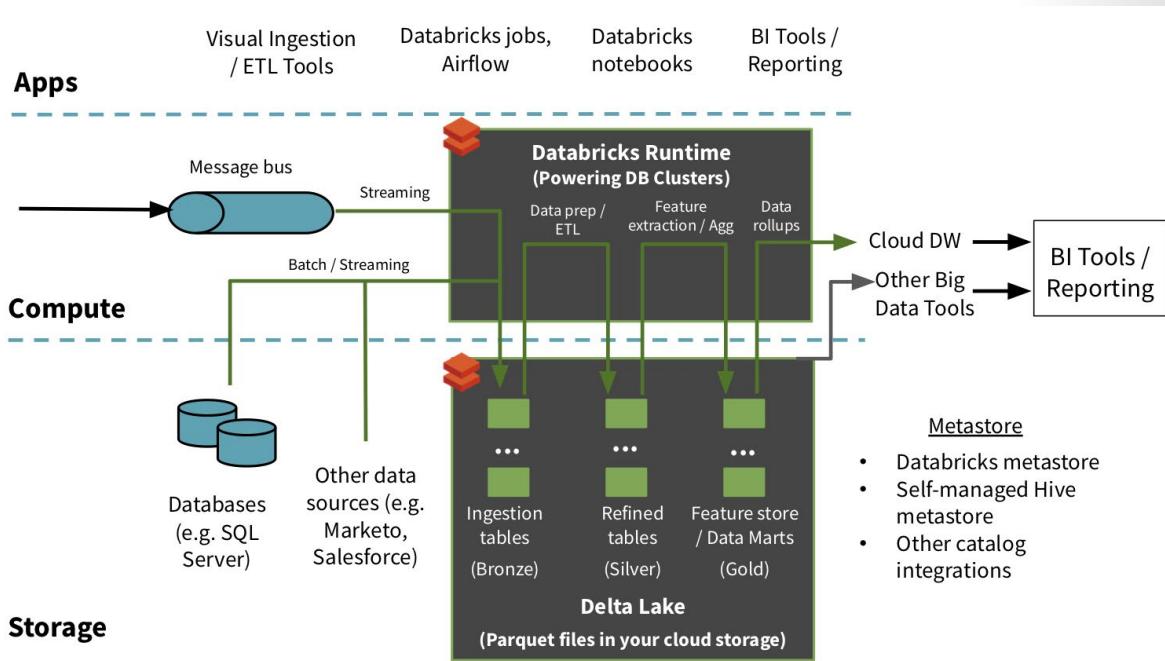
The Delta Lake Architecture is a vast improvement upon the traditional Lambda architecture. At each stage, we enrich our data through a unified pipeline that allows us to combine batch and streaming workflows through a shared filestore with ACID-compliant transactions.

Bronze tables contain raw data ingested from various sources (JSON files, RDBMS data, IoT data, etc.).

Silver tables will provide a more refined view of our data. We can join fields from various bronze tables to enrich streaming records, or update account statuses based on recent activity.

Gold tables provide business level aggregates often used for reporting and dashboarding. This would include aggregations such as daily active website users, weekly sales per store, or gross revenue per quarter by department.

The end outputs are actionable insights, dashboards, and reports of business metrics.



By considering our business logic at all steps of the extract-transform-load (ETL) pipeline, we can ensure that storage and compute costs are optimized by reducing unnecessary duplication of data and limiting ad hoc querying against full historic data.

Each stage can be configured as a batch or streaming job, and ACID transactions ensure that we succeed or fail completely.

Perform batch and stream processing

Now, let's jump into an Azure Databricks workspace and perform Structured Streaming with batch jobs by using Delta Lake.

In this unit, you need to complete the exercises within a Databricks Notebook. To begin, you need to have access to an Azure Databricks workspace. If you do not have a workspace available, follow the instructions below. Otherwise, you can skip to the bottom of the page to Clone the Databricks archive.

Unit pre-requisites

Microsoft Azure Account: You will need a valid and active Azure account for the Azure labs. If you do not have one, you can sign up for a [free trial](#)⁵

- If you are a Visual Studio Active Subscriber, you are entitled to Azure credits per month. You can refer to this [link](#)⁶ to find out more including how to activate and start using your monthly Azure credit.
- If you are not a Visual Studio Subscriber, you can sign up for the FREE **Visual Studio Dev Essentials**⁷ program to create Azure free account.

⁵ <https://azure.microsoft.com/free/>

⁶ <https://azure.microsoft.com/pricing/member-offers/msdn-benefits-details/>

⁷ <https://www.visualstudio.com/dev-essentials/>

Create the required resources

To complete this lab, you will need to deploy an Azure Databricks workspace in your Azure subscription.

Deploy an Azure Databricks workspace

1. Click the following button to open the Azure Resource Manager template in the Azure portal.
Deploy Databricks from the Azure Resource Manager Template⁸
2. Provide the required values to create your Azure Databricks workspace:
 - **Subscription:** Choose the Azure Subscription in which to deploy the workspace.
 - **Resource Group:** Leave at Create new and provide a name for the new resource group.
 - **Location:** Select a location near you for deployment. For the list of regions supported by Azure Databricks, see **Azure services available by region⁹**.
 - **Workspace Name:** Provide a name for your workspace.
 - **Pricing Tier:** Ensure premium is selected.
3. Accept the terms and conditions.
4. Select Purchase.
5. The workspace creation takes a few minutes. During workspace creation, the portal displays the Submitting deployment for Azure Databricks tile on the right side. You may need to scroll right on your dashboard to see the tile. There is also a progress bar displayed near the top of the screen. You can watch either area for progress.

Create a cluster

1. When your Azure Databricks workspace creation is complete, select the link to go to the resource.
2. Select **Launch Workspace** to open your Databricks workspace in a new tab.
3. In the left-hand menu of your Databricks workspace, select **Clusters**.
4. Select **Create Cluster** to add a new cluster.

⁸ <https://portal.azure.com/#create/Microsoft.Template/uri/https%3A%2F%2Fraw.githubusercontent.com%2FAzure%2Fazure-quickstart-templates%2Fmaster%2F101-databricks-workspace%2Fazuredeploy.json>

⁹ <https://azure.microsoft.com/regions/services/>

Create Cluster

New Cluster

[Cancel](#)

[Create Cluster](#)

2-8 Workers: 28.0-112.0 GB Memory, 8-32 Cores
1 Driver: 14.0 GB Memory, 4 Cores, 0.75 DBU

Cluster Name

Test Cluster



Cluster Mode

Standard

Pool

None

Databricks Runtime Version

[Learn more](#)

Runtime: 6.4 (Scala 2.11, Spark 2.4.5)

New This Runtime version supports only Python 3.

Autopilot Options

Enable autoscaling

Terminate after minutes of inactivity

Worker Type

Standard_DS3_v2

14.0 GB Memory, 4 Cores, 0.75 DBU

Min Workers

2

Max Workers

8

Driver Type

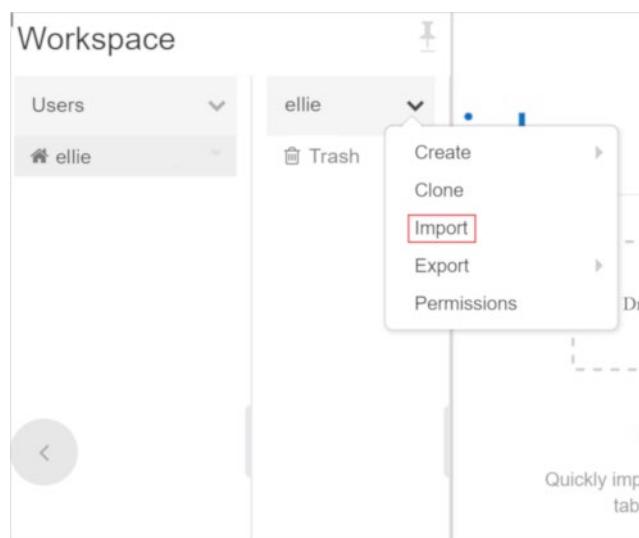
Same as worker

14.0 GB Memory, 4 Cores, 0.75 DBU

5. Enter a name for your cluster. Use your name or initials to easily differentiate your cluster from your coworkers.
6. Select the **Databricks RuntimeVersion**. We recommend the latest runtime and **Scala 2.11**.
7. Select the default values for the cluster configuration.
8. Select **Create Cluster**.

Clone the Databricks archive

1. If you do not currently have your Azure Databricks workspace open: in the Azure portal, navigate to your deployed Azure Databricks workspace and select **Launch Workspace**.
2. In the left pane, select **Workspace > Users**, and select your username (the entry with the house icon).
3. In the pane that appears, select the arrow next to your name, and select **Import**.



4. In the **Import Notebooks** dialog box, select the URL and paste in the following URL:

```
https://github.com/solliancenet/microsoft-learning-paths-databricks-notebooks/blob/master/data-engineering/DBC/11-Delta-Lake-Architecture dbc?raw=true
```

5. Select **Import**.
6. Select the **11-Delta-Lake-Architecture** folder that appears.

Complete the following notebook

Open the **1-Delta-Architecture** notebook. Make sure you attach your cluster to the notebook before following the instructions and running the cells within.

Within the notebook, you will explore combining streaming and batch processing with a single pipeline.

After you've completed the notebook, return to this screen, and continue to the next step.

Knowledge check

Question 1

What is a lambda architecture and what does it try to solve?

- An architecture that defines a data processing pipeline whereby microservices act as compute resources for efficient large-scale data processing
- An architecture that splits incoming data into two paths - a batch path and a streaming path. This architecture helps address the need to provide real-time processing in addition to slower batch computations.
- An architecture that employs the latest Scala runtimes in one or more Databricks clusters to provide the most efficient data processing platform available today

Question 2

What command should be issued to view the list of active streams?

- Invoke **spark.streams.active**
- Invoke **spark.streams.show**
- Invoke **spark.view.active**

Question 3

What is required to specify the location of a checkpoint directory when defining a Delta Lake streaming query?

- `.writeStream.format("delta").checkpoint("location", checkpointPath) ...`
- `.writeStream.format("delta").option("checkpointLocation", checkpointPath) ...`
- `.writeStream.format("parquet").option("checkpointLocation", checkpointPath) ...`

Summary

If you're looking for a data-management system that's fast, reliable, and able to handle large volumes of data in different raw formats, Databricks Delta is the solution. Databricks Delta provides the best of data lake, data warehousing, and streaming data-ingestion systems.

During this module, you explored combining streaming and batch processing with a single pipeline. Now you should know how to:

- Ingest streaming JSON data from disk and write it to a bronze Delta Lake Table.
- Perform a Stream-Static Join on the streamed data to add additional geographic data.
- Transform and load the data, saving it out to a silver Delta Lake Table.
- Summarize the data through aggregation into a gold Delta Lake Table.
- Materialize views of the gold table through streaming plots and static queries.
- Write batches of data back to the bronze table to trigger the same logic on newly loaded data and propagate your changes automatically.

Clean up

If you plan on completing other Azure Databricks modules, don't delete your Azure Databricks instance yet. You can use the same environment for the other modules.

Delete the Azure Databricks instance

1. Navigate to the Azure portal.
2. Navigate to the resource group that contains your Azure Databricks instance.
3. Select **Delete resource group**.
4. Type the name of the resource group in the confirmation text box.
5. Select **Delete**.

Introduction to Azure Data Lake storage

Introduction

Many organizations have spent the last two decades building data warehouses and business intelligence (BI) solutions based on relational database systems. Many BI solutions have lost out on opportunities to store unstructured data due to cost and complexity in these types of data and databases.

Suppose that you're a data engineering consultant doing work for Contoso. They're interested in using Azure to analyze all of their business data. In this role, you'll provide guidance on how Azure can enhance their existing business intelligence systems. You'll also offer advice about using Azure's storage capabilities to store large amounts of unstructured data to add value to their BI solution. Because of their needs, you plan to recommend Azure Data Lake Storage. Data Lake Storage provides a repository where you can upload and store huge amounts of unstructured data with an eye toward high-performance big data analytics.

Learning objectives

In this module you will:

- Learn about Azure Data Lake Storage
- Create an Azure Storage Account by using the Azure portal
- Compare Data Lake Storage Gen2 and Azure Blob Storage
- Explore the stages for processing big data by using Data Lake Storage
- Review the use cases for Data Lake Storage

Understand Azure Data Lake Storage Gen2

A data lake is a repository of data that is stored in its natural format, usually as blobs or files. Azure Data Lake Storage is a comprehensive, scalable, and cost-effective data lake solution for big data analytics built into Azure.



<https://channel9.msdn.com/Shows/Learn-Azure/Understanding-Azure-Data-Lake-Storage-Gen-2/player?format=ny>

Azure Data Lake Storage combines a file system with a storage platform to help you quickly identify insights into your data. Data Lake Storage Gen2 builds on Azure Blob storage capabilities to optimize it specifically for analytics workloads. This integration enables analytics performance, the tiering and data lifecycle management capabilities of Blob storage, and the high-availability, security, and durability capabilities of Azure Storage.

The variety and volume of data that is generated and analyzed today is increasing. Companies have multiple sources of data, from websites to Point of Sale (POS) systems, and more recently from social

media sites to Internet of Things (IoT) devices. Each source provides an essential aspect of data that needs to be collected, analyzed, and potentially acted upon.

Benefits

Data Lake Storage Gen2 is designed to deal with this variety and volume of data at exabyte scale while securely handling hundreds of gigabytes of throughput. With this, you can use Data Lake Storage Gen2 as the basis for both real-time and batch solutions. Here is a list of additional benefits that Data Lake Storage Gen2 brings:

Hadoop compatible access

A benefit of Data Lake Storage Gen2 is that you can treat the data as if it's stored in a Hadoop Distributed File System. With this feature, you can store the data in one place and access it through compute technologies including Azure Databricks, Azure HDInsight, and Azure Synapse Analytics without moving the data between environments.

Security

Data Lake Storage Gen2 supports access control lists (ACLs) and Portable Operating System Interface (POSIX) permissions. You can set permissions at a directory level or file level for the data stored within the data lake. This security is configurable through technologies such as Hive and Spark, or utilities such as Azure Storage Explorer. All data that is stored is encrypted at rest by using either Microsoft or customer-managed keys.

Performance

Azure Data Lake Storage organizes the stored data into a hierarchy of directories and subdirectories, much like a file system, for easier navigation. As a result, data processing requires less computational resources, reducing both the time and cost.

Data redundancy

Data Lake Storage Gen2 takes advantage of the Azure Blob replication models that provide data redundancy in a single data center with locally redundant storage (LRS), or to a secondary region by using the Geo-redundant storage (GRS) option. This feature ensures that your data is always available and protected if catastrophe strikes.

Exercise Create an Azure Storage Account by using the portal

Azure Data Lake Storage Gen2 is easy to set up. It requires a **StorageV2 (General Purpose V2)** Azure Storage account with the Hierarchical namespace enabled. Let's walk through an example of setting up an Data Lake Storage account in the Azure portal.

1. Sign in to the **Azure portal**¹⁰
2. Select **Create a resource** and in the textbox that states "Search the Marketplace type **Storage account**, and click on **Storage account**.

¹⁰ <https://portal.azure.com?azure-portal=true>

3. In **Storage account** screen, click **Create**.

Home >

Storage account

Microsoft

Storage account Microsoft Add to Favorites
★ ★ ★ ★ 4.2 (1730 ratings)
Azure benefit eligible ↗
Create

Overview Plans Usage Information + Support Reviews

Microsoft Azure provides scalable, durable cloud storage, backup, and recovery solutions for any data, big or small. It works with the infrastructure you already have to cost-effectively enhance your existing applications and business continuity strategy, and provide the storage required by your cloud applications, including unstructured text or binary data such as video, audio, and images.

4. Next, in the **Create storage account** window, in the **Basics** tab, under Project details section, ensure that your subscription is selected, and the appropriate resource group. Under the Instance section, define a **storage account name**. Set the **Region** to **Central US**. In the **Performance*** radio button list, select **Standard**, and set the Redundancy to **Locally-redundant storage (LRS)**.

The screenshot shows the 'Create a storage account' wizard in the Azure portal. The 'Basics' tab is selected. At the top, there's a breadcrumb navigation: Home > Storage account > Create a storage account. Below the tabs, a descriptive text explains Azure Storage as a Microsoft-managed service providing cloud storage. It mentions Azure Blobs, Azure Data Lake Storage Gen2, Azure Files, Azure Queues, and Azure Tables, noting that cost depends on usage and options.

Project details

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: ctodbrg (dropdown menu)

Resource group: ctodbrg (dropdown menu) | Create new

Instance details

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

Storage account name: ctostorageacc21

Region: (US) Central US (dropdown menu)

Performance: Standard (selected) | Premium (radio buttons)

Redundancy: Locally-redundant storage (LRS) (dropdown menu)

5. Select the **Advanced** tab. Under the section Data Lake Storage Gen2, click the checkbox next to **Enable hierarchical namespace**, as shown below.

Home > Storage account >
Create a storage account ...

Basics **Advanced** Networking Data protection Tags Review + create

i Certain options have been disabled by default due to the combination of storage account performance, redundancy, and region.

Security

Configure security settings that impact your storage account.

Enable secure transfer i

Enable infrastructure encryption i
i Sign up is currently required to enable infrastructure encryption on a per-subscription basis. [Sign up](#)

Enable blob public access i

Enable storage account key access i

Minimum TLS version i ▼

Data Lake Storage Gen2

The Data Lake Storage Gen2 hierarchical namespace accelerates big data analytics workloads and enables file-level access control lists (ACLs). [Learn more](#)

Enable hierarchical namespace

Blob storage

Enable network file share v3 i
i Sign up is currently required to enable NFS v3 on a per-subscription basis. [Sign up](#)

Access tier i Hot: Frequently accessed data and day-to-day usage scenarios
—

6. Select the **Review + create** tab, and click **Create**.

This new Azure Storage account is now set up to host data for an Azure Data Lake. After the account has deployed, you will find options related to Azure Data Lake in the Overview page.

Compare Azure Data Lake Store to Azure Blob storage

In Azure Blob storage, you can store large amounts of unstructured ("object") data, in a single hierarchy, also known as a flat namespace. You can access this data by using HTTP or HTTPS. Azure Data Lake Storage Gen2 builds on blob storage and optimizes I/O of high-volume data by using hierarchical namespaces that you turned on in the previous exercise.

Hierarchical namespaces organize blob data into *directories* and stores metadata about each directory and the files within it. This structure allows operations, such as directory renames and deletes, to be performed in a single atomic operation. Flat namespaces, by contrast, require several operations proportionate to the number of objects in the structure. Hierarchical namespaces keep the data organized, which yields better storage and retrieval performance for an analytical use case and lowers the cost of analysis.

Azure Blob storage vs. Azure Data Lake Storage

If you want to store data *without performing analysis on the data*, set the **Hierarchical Namespace** option to **Disabled** to set up the storage account as an Azure Blob storage account. You can also use blob storage to archive rarely used data or to store website assets such as images and media.

If you are performing analytics on the data, set up the storage account as an Azure Data Lake Storage Gen2 account by setting the **Hierarchical Namespace** option to **Enabled**. Because Azure Data Lake Storage Gen2 is integrated into the Azure Storage platform, applications can use either the Blob APIs or the Azure Data Lake Storage Gen2 file system APIs to access data.

Understand the stages for processing big data by using Azure Data Lake Store

Azure Data Lake Storage Gen2 plays a fundamental role in a wide range of big data architectures. These architectures can involve the creation of:

- A modern data warehouse.
- Advanced analytics against big data.
- A real-time analytical solution.

There are four stages for processing big data solutions that are common to all architectures:

- **Ingestion** - The ingestion phase identifies the technology and processes that are used to acquire the source data. This data can come from files, logs, and other types of unstructured data that must be put into the Data Lake Store. The technology that is used will vary depending on the frequency that the data is transferred. For example, for batch movement of data, Azure Data Factory may be the most appropriate technology to use. For real-time ingestion of data, Apache Kafka for HDInsight or Stream Analytics may be an appropriate technology to use.
- **Store** - The store phase identifies where the ingested data should be placed. In this case, we're using Azure Data Lake Storage Gen2.
- **Prep and train** - The prep and train phase identifies the technologies that are used to perform data preparation and model training and scoring for data science solutions. The common technologies that are used in this phase are Azure Databricks, Azure HDInsight or Azure Machine Learning Services.

- **Model and serve** - Finally, the model and serve phase involves the technologies that will present the data to users. These can include visualization tools such as Power BI, or other data stores such as Azure Synapse Analytics, Azure Cosmos DB, Azure SQL Database, or Azure Analysis Services. Often, a combination of these technologies will be used depending on the business requirements.

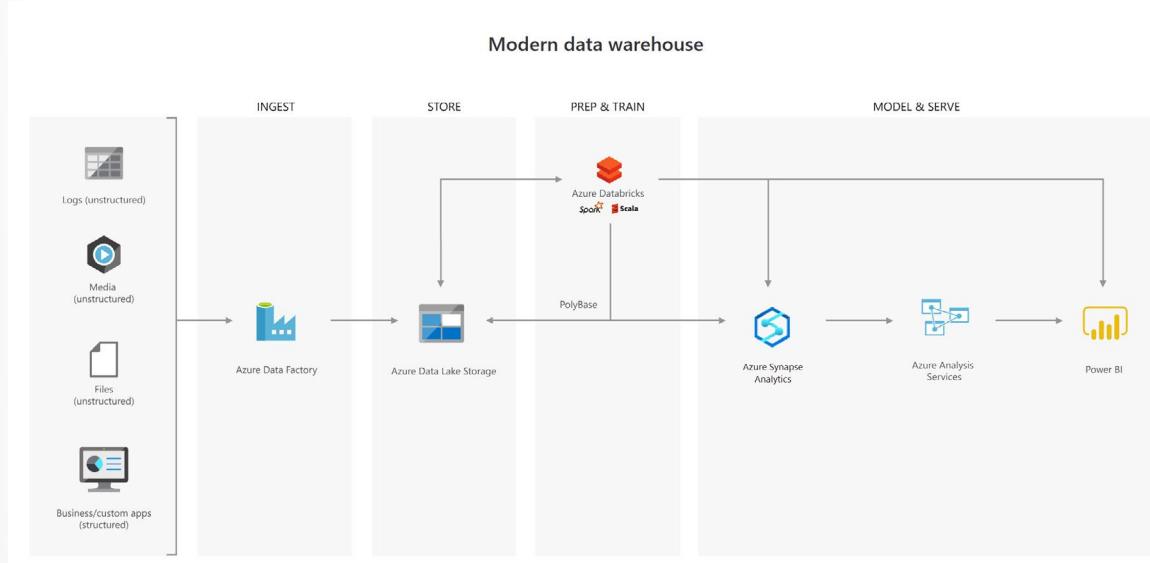
Examine uses for Azure Data Lake Storage Gen2

Let's examine three use cases for using an Azure Data Lake Store.

Creating a modern data warehouse

Imagine you're a Data Engineering consultant for Contoso. In the past, they've created an on-premises business intelligence solution that used a Microsoft SQL Server Database Engine, SQL Server Integration Services, SQL Server Analysis Services, and SQL Server Reporting Services to provide historical reports. They tried using the Analysis Services Data Mining component to create a predictive analytics solution to predict the buying behavior of customers. While this approach worked well with low volumes of data, it couldn't scale after more than a gigabyte of data was collected. Furthermore, they were never able to deal with the JSON data that a third-party application generated when a customer used the feedback module of the point of sale (POS) application.

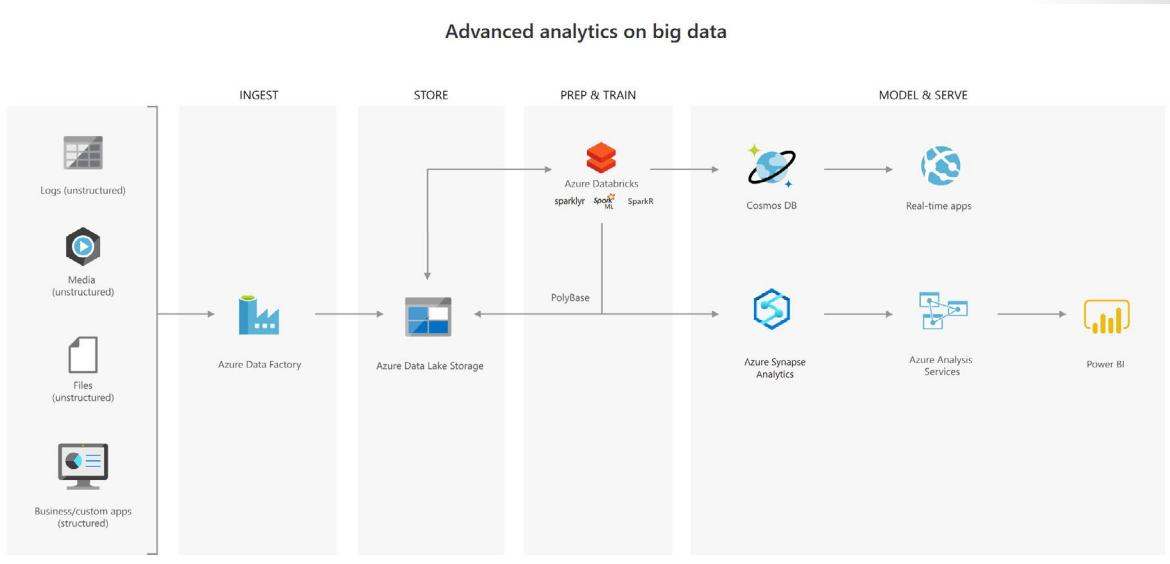
Contoso has turned to you for help with creating an architecture that can scale with the data needs that are required to create a predictive model and to handle the JSON data so that it's integrated into the BI solution. You suggest the following architecture:



The architecture uses Azure Data Lake Storage at the center of the solution for a modern data warehouse. Integration Services is replaced by Azure Data Factory to ingest data into the Data Lake from a business application. This is the source for the predictive model that is built into Azure Databricks. PolyBase is used to transfer the historical data into a big data relational format that is held in Azure Synapse Analytics, which also stores the results of the trained model from Databricks. Azure Analysis Services provides the caching capability for SQL Data Warehouse to service many users and to present the data through Power BI reports.

Advanced analytics for big data

In this second use case, Azure Data Lake Storage plays an important role in providing a large-scale data store. Your skills are needed by AdventureWorks, which is a global seller of bicycles and cycling components through a chain of resellers and on the internet. As their customers browse the product catalog on their websites and add items to their baskets, a recommendation engine that is built into Azure Databricks recommends other products. They need to make sure that the results of their recommendation engine can scale globally. The recommendations are based on the web log files that are stored on the web servers and transferred to the Azure Databricks model hourly. The response time for the recommendation should be less than 1 ms. You propose the following architecture:



In this solution, Azure Data Factory transfers terabytes of web logs from a web server to the Azure Data Lake on an hourly basis. This data is provided as features to the predictive model in Azure Databricks, which is then trained and scored. The results are distributed globally by using Azure Cosmos DB, which the real-time app (the AdventureWorks website) will use to provide recommendations to customers as they add products to their online baskets.

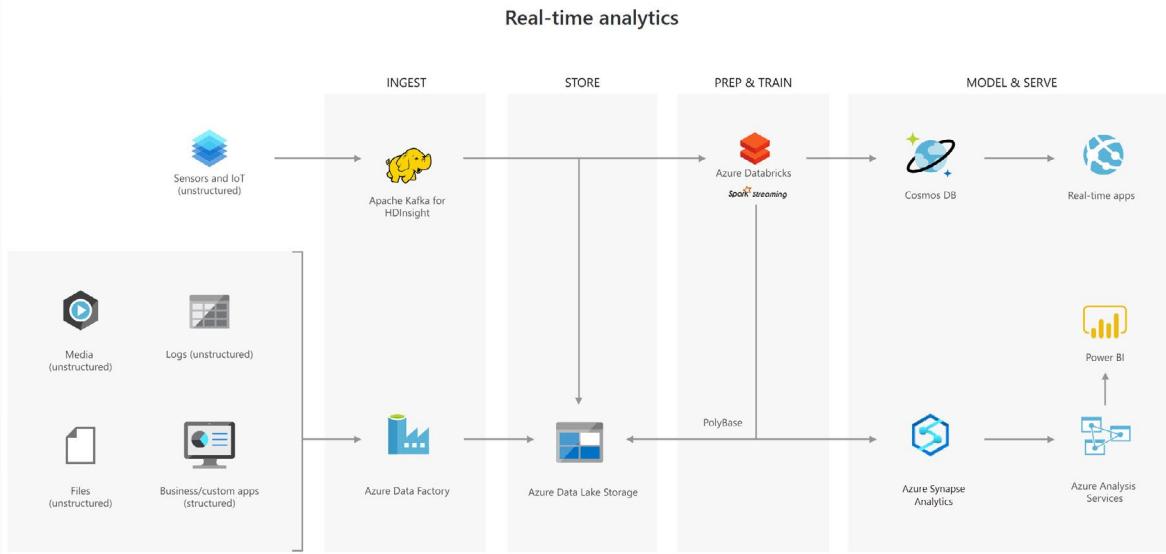
To complete this architecture, PolyBase is used against the Data Lake to transfer descriptive data to the SQL Data Warehouse for reporting purposes. Azure Analysis Services provides the caching capability for SQL Data Warehouse to service many users and to display the data through Power BI reports.

Real-time analytical solutions

To perform real-time analytical solutions, the ingestion phase of the architecture is changed for processing big data solutions. In this architecture, note the introduction of Apache Kafka for Azure HDInsight to ingest streaming data from an Internet of Things (IoT) device, although this could be replaced with Azure IoT Hub and Azure Stream Analytics. The key point is that the data is persisted in Data Lake Storage Gen2 to service other parts of the solution.

In this use case, you are a Data Engineer for Trey Research, an organization that is working with a transport company to monitor the fleet of Heavy Goods Vehicles (HGV) that drive around Europe. Each HGV is equipped with sensor hardware that will continuously report metric data on the temperature, the speed, and the oil and brake solution levels of an HGV. When the engine is turned off, the sensor also outputs a file with summary information about a trip, including the mileage and elevation of a trip. A trip is a period in which the HGV engine is turned on and off.

Both the real-time data and batch data is processed in a machine learning model to predict a maintenance schedule for each of the HGVs. This data is made available to the downstream application that third-party garage companies can use if an HGV breaks down anywhere in Europe. In addition, historical reports about the HGV should be visually presented to users. As a result, the following architecture is proposed:



In this architecture, there are two ingestion streams. Azure Data Factory ingests the summary files that are generated when the HGV engine is turned off. Apache Kafka provides the real-time ingestion engine for the telemetry data. Both data streams are stored in Azure Data Lake Store for use in the future, but they are also passed on to other technologies to meet business needs. Both streaming and batch data are provided to the predictive model in Azure Databricks, and the results are published to Azure Cosmos DB to be used by the third-party garages. PolyBase transfers data from the Data Lake Store into SQL Data Warehouse where Azure Analysis Services creates the HGV reports by using Power BI.

Knowledge check

Question 1

Mike is creating an Azure Data Lake Storage Gen2 account. He must configure this account to be able to process analytical data workloads for best performance. Which option should he configure when creating the storage account?

- On the **Basic** tab, set the **Performance** option to **Standard**.
- On the **Basic Tab**, set the **Performance** option to **ON**.
- On the **Advanced** tab, set the **Hierarchical Namespace** to **Enabled**.

Question 2

In which phase of big data processing is Azure Data Lake Storage located?

- Ingestion
- Store
- Model & Serve

Summary

Azure Data Lake Storage Gen2 provides a cloud storage service that is available, secure, durable, scalable, and redundant. It's a comprehensive data lake solution.

Azure Data Lake Storage brings new efficiencies to process big data analytics workloads and can provide data to many compute technologies including Azure HDInsight and Azure Databricks without needing to move the data around. Creating an Azure Data Lake Storage Gen2 data store can be an important tool in building a big data analytics solution.

Work with data streams by using Azure Stream Analytics (Repo content is updated)

Introduction

Today, massive amounts of real-time data are generated by connected applications, Internet of Things (IoT) devices and sensors, and various other sources. The proliferation of these streaming data sources has made the ability to consume and make informed decisions from these data in near-real-time an operational necessity for many organizations.

To provide a few examples, online stores analyze real-time clickstream data to provide product recommendations to consumers as they browse the website. Manufacturing facilities utilize telemetry data from IoT sensors to remotely monitor high-value assets. And credit card transactions from point-of-sale systems are scrutinized in real-time to detect and prevent potentially fraudulent activities.

Azure Stream Analytics¹¹ seamlessly integrates your real-time application architecture with a streaming analytics engine to transform streaming data into actionable insights. Using Azure Stream Analytics enables powerful, real-time analytics on your data no matter what the volume.

In this module, you will learn how real-time analytics of streaming data works, in principle. You will also discover how you can use Azure Stream Analytics to integrate streaming data into your real-time analytics workflows.

Learning objectives

In this module, you will:

- Understand data streams
- Understand event processing
- Learn about processing events with Azure Stream Analytics

Understand data streams

In the context of analytics, data streams are the data pertaining to the occurrence of specific activities that are emitted by applications, IoT devices or sensors, or other sources known as data producers. These perpetually generated data streams typically contain temporal and additional information about the events. The proliferation of connected applications and devices has led to exponential growth in the number of streaming data sources in recent years.

Data streams are most often used to better understand change over time. For example, an organization may perform sentiment analysis on tweets to see if an advertising campaign results in more positive comments about the company or its products.

Analyzing data streams

In today's world, data streams are ubiquitous. Analyzing a data stream is typically performed to measure how an event's state changes over time or to capture information on an area of interest. The intent being to:

- Continuously analyze data to detect issues and understand or respond to them

¹¹ <https://docs.microsoft.com/azure/stream-analytics/stream-analytics-introduction>

- Understand component or system behavior under various conditions to fuel further enhancements of that component or system
- Trigger specific actions or alerts when certain thresholds are hit

By analyzing data streams and extracting actionable insights, companies can harness latent knowledge to improve efficiencies, further innovation, and respond to irregularities. Examples of use cases that analyze data streams include:

<ul style="list-style-type: none"> • Stock market trends • Monitoring data of water pipelines and electrical transmission and distribution systems by utility companies • Mechanical component health monitoring data in automotive and automobile industries • Monitoring data from industrial and manufacturing equipment 	<ul style="list-style-type: none"> • Sensor data in transportation, such as traffic management and highway toll lanes • Patient health monitoring data in the healthcare industry • Satellite data in the space industry • Fraud detection in the banking and finance industries • Sentiment analysis of social media posts
---	--

Approaches to data stream processing

There are two approaches to processing data streams: live and on-demand.

The most commonly adopted method for processing data streams is to analyze new data continuously as it arrives from an event producer, such as Azure Event Hubs. This “live” approach requires more processing power to run computations but offers the ability to gain near-real-time insights. Using a service like Azure Stream Analytics, you can execute calculations and aggregations against arriving data using temporal analysis. The results of those queries can be sent to a Power BI dashboard for real-time visualization and analysis.

The diagram below depicts an end-to-end “live” data stream processing solution using Event Hubs to ingest streaming data, Azure Stream Analytics to transform data, and Power BI to visualize and analyze it.

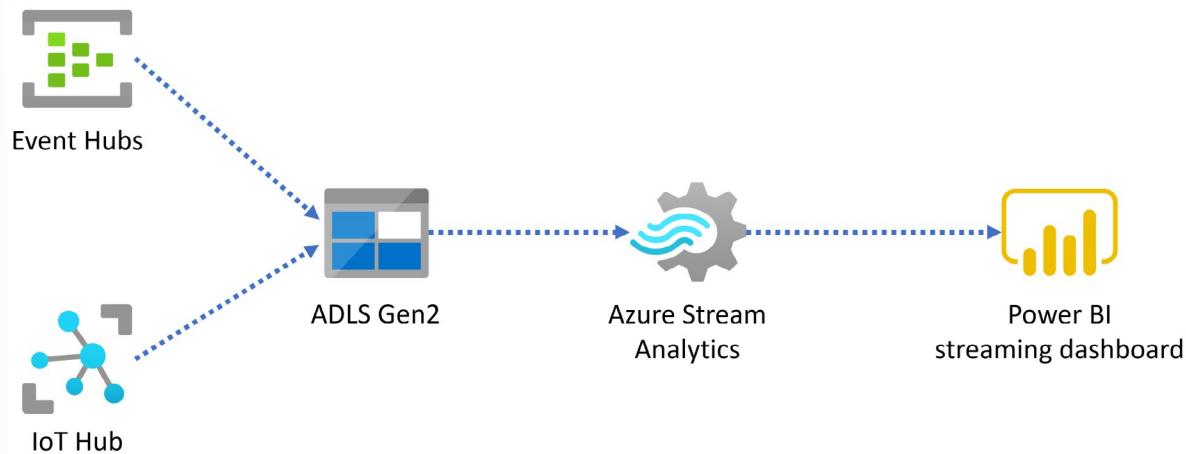


The “on-demand” approach for processing streaming data involves persisting all incoming data in a data store, such as **Azure Data Lake Storage (ADLS) Gen2**¹². This method allows you to collect streaming data over time and store it as static data. You can then process the *static* data in batches when convenient or during times when compute costs are lower.

The following diagram illustrates an on-demand data stream processing solution. Streaming data from Azure Event Hubs and IoT Hub are written as blobs into Azure Data Lake Storage (ADLS) Gen2. The static

¹² <https://docs.microsoft.com/azure/storage/blobs/data-lake-storage-introduction>

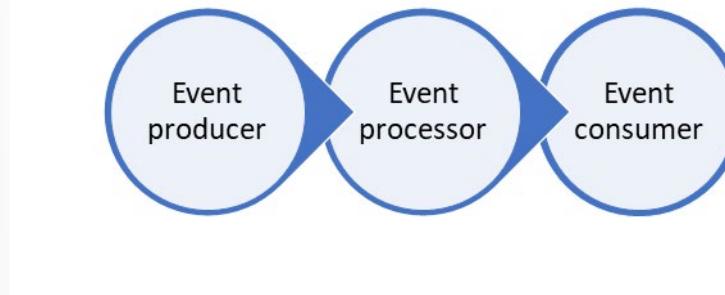
data are then processed using Azure Stream Analytics and output to a Power BI dashboard for visualization and analysis.



Understand event processing

Event processing refers to the consumption and analysis of a continuous data stream to extract latent knowledge and derive actionable insights from the events happening within that stream. Event processing pipelines provide an end-to-end solution for ingesting, transforming, and analyzing data streams and are made up of three distinct components:

- An **event producer**, which generates an event data stream
- An **event processor** responsible for the ingestion and transformation of streaming event data
- An **event consumer** that displays or consumes event data and takes action on it



Event producer

An event producer is an application, connected device or sensor, or any other service that continuously outputs a data stream of events. Events can be any recurring action, such as a heartbeat, a car passing through a toll booth, or an engine sensor reporting temperature values on an automobile. Every event should include temporal information, such as a timestamp, indicating when it occurred.

Azure Event Hubs and IoT Hub are frequently used as event producers in Azure. These services can handle incoming events at a massive scale and produce an event stream to feed into downstream event processing services, such as Azure Stream Analytics.

Event processor

An event processor is an engine designed to consume and transform event data streams. Event processors require the ability to query time segments easily. Performing time-boxed computations or aggregations, such as counting the number of times an event happens during a particular period, is a frequent use case. Depending on the problem space, event processors either process one incoming event at a time, such as a heart rate monitor, or process multiple events simultaneously, such as Azure Stream Analytics processing sensor data from highway tollbooths.

Azure Stream Analytics¹³ is the quickest way to get event processing running on Azure. Using Stream Analytics, you can ingest streaming data from Azure Events Hubs or IoT Hub and run real-time analytics queries against the streams.

Stream Analytics supports multiple types of windowing functions for performing temporal computations on data streams, providing a way to aggregate events over various time intervals depending on specific window definitions. It also provides the capability to use Azure Machine Learning functions to make it a robust tool for analyzing data streams.

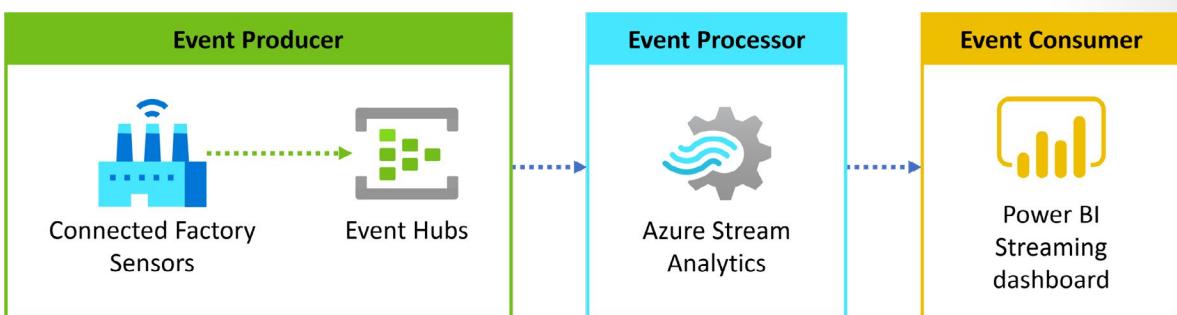
Event consumer

An event consumer is an application that consumes the output of an event processor. Event consumers can be used to visualize data or take a specific action based on the insights, such as generating alerts when specified thresholds are met or sending data to another event processing engine.

For visualizing and analyzing data, **Power BI**¹⁴ is the recommended event consumer. It provides a platform for creating complex linked visualizations and analyzing aggregated data in near-real-time. Azure Stream Analytics can output directly to Power BI, allowing dashboards to be updated continuously as data streams are processed.

Building event processing pipelines

Event processing pipelines generally chain together multiple services to create a near-real-time analytics pipeline. The diagram below shows an example event processing pipeline built on Event Hubs, Azure Stream Analytics, and Power BI.



In this pipeline, Event Hubs ingests streaming data from sensors in a connected factory, and together they act as the event producer. The event processor is Azure Stream Analytics, which receives the data from Event Hubs and transforms and aggregates it. Power BI, the event consumer, receives output from Stream Analytics and displays rich visualizations used to analyze the event data.

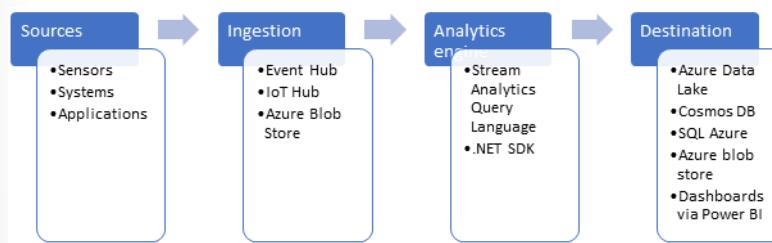
¹³ <https://docs.microsoft.com/azure/stream-analytics/stream-analytics-introduction>

¹⁴ <https://docs.microsoft.com/power-bi/fundamentals/power-bi-overview>

Process events azure stream analytics

Azure Stream Analytics is a platform-as-a-service (PaaS) event processing engine. It enables the transformation and analysis of large volumes of streaming data arriving from Azure Event Hubs and IoT Hub and static data from Azure storage. Using Stream Analytics, you can write complex time-based queries and aggregations over the data generated by connected sensors, devices, or applications. Stream Analytics processes the data in real-time, enabling powerful insights to drive real-time decision-making. A typical event processing pipeline built on top of Stream Analytics consists of the following four components:

- **Event producer:** Any application, system, or sensor that continuously produces event data of interest. Examples include sensors tracking the flow of water through a utility pipe and an application such as Twitter that generates tweets against a single hashtag.
- **Event ingestion system:** Receives the data from an event producer and passes it to an analytics engine. Azure Event Hubs, Azure IoT Hub, or Azure Blob storage can serve as the ingestion system.
- **Stream analytics engine:** The compute platform that processes, aggregates, and transforms incoming data streams. Azure Stream Analytics provides the Stream Analytics query language (SAQL), a subset of Transact-SQL tailored to perform computations over streaming data. The engine supports windowing functions that are fundamental to stream processing and are implemented by using the SAQL.
- **Event consumer:** A destination of the output from the stream analytics engine. The output can be stored in a data storage platform, such as Azure Data Lake Storage Gen2, Azure Cosmos DB, Azure SQL Database, or Azure Blob storage. Or, you can consume the output in near-real-time using Power BI dashboards.



Operational aspects

Stream Analytics guarantees *exactly once* event processing and *at-least-once* event delivery, so events are never lost. It has built-in recovery capabilities in case the delivery of an event fails. Also, Stream Analytics provides built-in checkpointing to maintain the state of your job and produces repeatable results.

Because Azure Stream Analytics is a PaaS service, it's fully managed and highly reliable. Its built-in integration with various sources and destinations and flexible programmability model enhance programmer productivity. The Stream Analytics engine enables in-memory compute, so it offers superior performance. All these factors contribute to the low total cost of ownership (TCO) of Azure Stream Analytics.

Knowledge check

Question 1

Which of the following technologies typically provide an ingestion point for data streaming in an event processing solution that uses static data as a source?

- Azure IoT Hub
- Azure Blob storage
- Azure Event Hub

Question 2

To consume processed event streaming data in near-real-time to produce dashboards containing rich visualizations, which of the following services should you use?

- Azure Cosmos DB
- Event Hubs
- Power BI

Summary

Azure Stream Analytics is a PaaS service that integrates with your applications and Internet of Things (IoT) to gain insights with streaming data or static data held in a blob store. The process of consuming data streams, analyzing them, and deriving actionable insights out of them is called event processing. It requires an event producer, an event processor, and an event consumer. Azure Stream Analytics provides the event processing aspect to streaming that's fully managed and highly reliable.

In this module, you learned how real-time analytics of streaming data works, in principle. You also discovered how you can use Azure Stream Analytics to integrate streaming data into your real-time analytics workflows.

Learning objectives

In this module, you have:

- Understood data streams
- Understood event processing
- Learned about processing events with Azure Stream Analytics

Module Lab information

Lab 1 - Explore compute and storage options for data engineering workloads

- **Estimated Time:** 50 - 60 minutes
- **Lab files:** The files for this lab are located in the *Instructions\Labs\01* folder.

Lab overview

This lab teaches ways to structure the data lake, and to optimize the files for exploration, streaming, and batch workloads. The student will learn how to organize the data lake into levels of data refinement as they transform files through batch and stream processing. The students will also experience working with Apache Spark in Azure Synapse Analytics. They will learn how to create indexes on their datasets, such as CSV, JSON, and Parquet files, and use them for potential query and workload acceleration using Spark libraries including Hyperspace and MSSparkUtils.

Lab objectives

After completing this lab, you will be able to:

- Perform Data Combine streaming and batch processing with a single pipeline
- Organize the data lake into levels of file transformation
- Index data lake storage for query and workload acceleration

Module summary

module-summary

In this module, you have learned how to get familiar with core compute technologies used for data engineering in Azure. You gained an understanding of the role that a data lake plays in providing a storage layer. You understand how to organize your data in a data lake to optimize for exploration, streaming and batch workloads.

Learning objectives

In this module, you have learned to:

- Understand Azure Synapse Analytics
- Describe Azure Databricks
- Describe Azure Databricks Delta Lake architecture
- Understand Azure Data Lake storage
- Work with data streams by using Azure Stream Analytics

Post Course Review

After the course, consider visiting [the Microsoft Customer Case Study site¹⁵](#). Use the search bar to search by an industry such as healthcare or retail, or by a technology such as Azure Synapse Analytics or Azure Databricks. Read through some of the customers stories.

Important

Remember

Should you be working in your own subscription while running through this content, it is best practice at the end of a project to identify whether or not you still need the resources you created. Resources left running can cost you money.

You can delete resources one by one, or just delete the resource group to get rid of the entire set.

¹⁵ <https://customers.microsoft.com/>

Answers

Question 1

Which type of analytics answers the question "What is likely to happen in the future based on previous trends and patterns?"

- Descriptive.
- Diagnostic.
- Predictive.

Explanation

Correct. Predictive analytics "What is likely to happen in the future based on previous trends and patterns?".

Question 2

You have a requirement to occasionally prepare data for ad hoc data exploration and analysis. Which resource model in Azure Synapse Analytics is the most effective to use to meet this requirement?

- Serverless.
- Dedicated.
- Pipelines.

Explanation

Correct. The serverless resource model is the ideal resource model in this scenario as it makes use of the resources when required.

Question 3

Where can you develop TSQL scripts and notebooks in Azure Synapse Analytics?

- Azure portal.
- Azure Synapse Studio.
- Data Lake.

Explanation

Correct. Azure Synapse Studio is where you can develop TSQL scripts and notebooks.

Question 1

How many drivers does a Cluster have?

- Only one
- Two, running in parallel
- Configurable between one and eight

Explanation

Correct. A Cluster has one and only one driver.

Question 2

Spark is a distributed computing environment. Therefore, work is parallelized across executors. At which two levels does this parallelization occur?

- The Executor and the Slot
- The Driver and the Executor
- The Slot and the Task

Explanation

Correct. The first level of parallelization is the Executor - a Java virtual machine running on a node, typically, one instance per node. Each Executor has a number of Slots to which parallelized Tasks can be assigned to it by the Driver.

Question 3

What type of process are the driver and the executors?

- Java processes
- Python processes
- C++ processes

Explanation

Correct. The driver and the executors are Java processes.

Question 1

What is a lambda architecture and what does it try to solve?

- An architecture that defines a data processing pipeline whereby microservices act as compute resources for efficient large-scale data processing
- An architecture that splits incoming data into two paths - a batch path and a streaming path. This architecture helps address the need to provide real-time processing in addition to slower batch computations.
- An architecture that employs the latest Scala runtimes in one or more Databricks clusters to provide the most efficient data processing platform available today

Explanation

Correct. The lambda architecture is a big data processing architecture that combines both batch- and real-time processing methods.

Question 2

What command should be issued to view the list of active streams?

- Invoke **spark.streams.active**
- Invoke **spark.streams.show**
- Invoke **spark.view.active**

Explanation

Correct. That's the correct syntax to view the list of active streams.

Question 3

What is required to specify the location of a checkpoint directory when defining a Delta Lake streaming query?

- `.writeStream.format("delta").checkpoint("location", checkpointPath) ...`
- `.writeStream.format("delta").option("checkpointLocation", checkpointPath) ...`
- `.writeStream.format("parquet").option("checkpointLocation", checkpointPath) ...`

Explanation

Correct. That's the correct syntax to specify the checkpoint directory on a Delta Lake streaming query.

Question 1

Mike is creating an Azure Data Lake Storage Gen2 account. He must configure this account to be able to process analytical data workloads for best performance. Which option should he configure when creating the storage account?

- On the **Basic** tab, set the **Performance** option to **Standard**.
- On the **Basic** Tab, set the **Performance** option to **ON**.
- On the **Advanced** tab, set the **Hierarchical Namespace** to **Enabled**.

Explanation

*Correct. If you want to enable the best performance for analytical workloads in Data Lake Storage Gen2, then on the **Advanced** tab of the Storage Account creation set the **Hierarchical Namespace** to **Enabled**.*

Question 2

In which phase of big data processing is Azure Data Lake Storage located?

- Ingestion
- Store
- Model & Serve

Explanation

Correct. Store is the phase in which Azure Data Lake Storage resides for processing big data solution.

Question 1

Which of the following technologies typically provide an ingestion point for data streaming in an event processing solution that uses *static* data as a source?

- Azure IoT Hub
- Azure Blob storage
- Azure Event Hub

Explanation

That's the correct answer. Azure Blob storage provides an ingestion point for data streaming in an event processing solution that uses static data as a source.

Question 2

To consume processed event streaming data in near-real-time to produce dashboards containing rich visualizations, which of the following services should you use?

- Azure Cosmos DB
- Event Hubs
- Power BI

Explanation

Correct. Power BI provides a platform for visualizing and analyzing aggregated data in near-real-time. Azure Stream Analytics can target Power BI as an output destination. Processed data is passed into Power BI to facilitate near-real-time dashboard updates.

Module 2 Run interactive queries using serverless SQL pools

Module introduction

module-introduction

In this module, you will learn how to query and prepare data in an interactive way on files placed in Azure Storage using Azure Synapse Analytics.

Learning objectives

In this module, you will:

- Explore Azure Synapse serverless SQL pools capabilities
- Query data in the lake using Azure Synapse serverless SQL pools
- Create metadata objects in Azure Synapse serverless SQL pools
- Secure data and manage users in Azure Synapse serverless SQL pools

Explore Azure Synapse serverless SQL pools capabilities

Introduction

In this lesson, you will explore the capabilities of Azure Synapse serverless SQL pools and how it can be used to interactively work with data in a data lake.

After the completion of this lesson, you will be able to:

- Identify Azure Synapse serverless SQL pools
- Describe when to use Azure Synapse serverless SQL pools
- Discuss Azure Synapse serverless SQL pools use cases

Prerequisites

Before taking this lesson, it is recommended that the student is able to:

- Log into the Azure portal
- Explain the different components of Azure Synapse Analytics
- Use Azure Synapse Studio

What is Azure Synapse serverless SQL pools

Azure Synapse Analytics is an integrated analytics service that accelerates time to insight across data warehouses and big data analytics systems. At its core, Azure Synapse brings together the best of SQL technologies used in enterprise data warehousing, Spark technologies used for big data, and Pipelines for data integration with extract, transform and load or extract, load, and transform (ETL/ELT) frameworks.

Azure Synapse Analytics has a web-based studio that provides a single place for management, monitoring, coding, and security. Azure Synapse Analytics features deep integration with other Azure services such as Power BI, CosmosDB, and AzureML.

Azure Synapse SQL is a distributed query system that enables enterprises to implement data warehousing and data virtualization scenarios using standard T-SQL experiences familiar to data engineers. It also expands the capabilities of SQL to address streaming and machine learning scenarios.

Azure Synapse SQL offers both serverless and dedicated resource models, offering consumption and billing options to fit your needs.

Every Azure Synapse Analytics workspace comes with built-in serverless SQL pool that you can use to query data in the lake.

The screenshot shows the Azure portal interface for a Synapse workspace named 'synapsectows'. The left sidebar contains navigation links for Home, Overview, Activity log, Access control (IAM), Tags, Settings, SQL Active Directory admin, Properties, Locks, Analytics pools (with SQL pools and Apache Spark pools listed), Security (Firewalls and Managed identities), Private endpoint connections, Monitoring (Alerts and Metrics), Automation (Tasks), Support + troubleshooting (New support request), and a search bar at the top.

Analytics pools

Name	Type	Size
Built-in	Serverless	Auto
SQLPool01	Dedicated	DW100c
Sparkpool01	Apache Spark pool	Small

Serverless SQL pool provides a pay per query endpoint to query the data in your data lake. It enables you to access your data through the following functionalities:

- A familiar T-SQL syntax to query data in place without the need to copy or load data into a specialized store.
- Integrated connectivity via the T-SQL interface that offers a wide range of business intelligence and ad-hoc querying tools, including the most popular drivers.

Serverless SQL pool is a distributed data processing system, built for large-scale data, and computational functions. It enables you to analyze your Big Data in seconds to minutes, depending on the workload. Thanks to built-in query execution fault-tolerance, the system provides high reliability and success rates even for long-running queries involving large data sets.

With serverless SQL pool, there's no infrastructure to setup or clusters to maintain. A built-in endpoint for this service is provided within every Azure Synapse workspace, so you can start querying data as soon as the workspace is created.

There is no charge for resources reserved, you are only being charged for the data processed by queries you run, hence this model is a true pay-per-use model.

When to use Azure Synapse serverless SQL pools

Synapse SQL offers both serverless and dedicated resource models, offering consumption and billing options to fit your needs. For predictable performance and cost, create dedicated SQL pools to reserve processing power for data stored in SQL tables. For unplanned or bursty workloads, use the always available, serverless SQL endpoint.

Synapse SQL serverless resource model is great if you need to know exact cost for each query executed to monitor and attribute costs. Additionally, it eliminates the management overhead since there is no

infrastructure to manage. You just care about the queries you want to execute. For these reasons getting the project started with serverless is simple and cheap.

Additionally, serverless SQL pool is tailored for querying the data residing in the data lake, so in addition to eliminating management burden, it eliminates a need to worry about ingesting the data into the system. You just point the query to the data that is already in the lake and run it.

Azure Synapse serverless SQL pools use cases

Serverless SQL pool acts and behaves like a regular SQL Server. So, all clients that can connect to SQL Server can connect to serverless SQL pool as well. This enables a plethora of use cases. In general, serverless SQL pool is an analytics system and it's not recommended to be used for OLTP type of workloads. Workloads that require millisecond response times and are looking to pinpoint a single row in a data set are not good fit for serverless SQL pool.

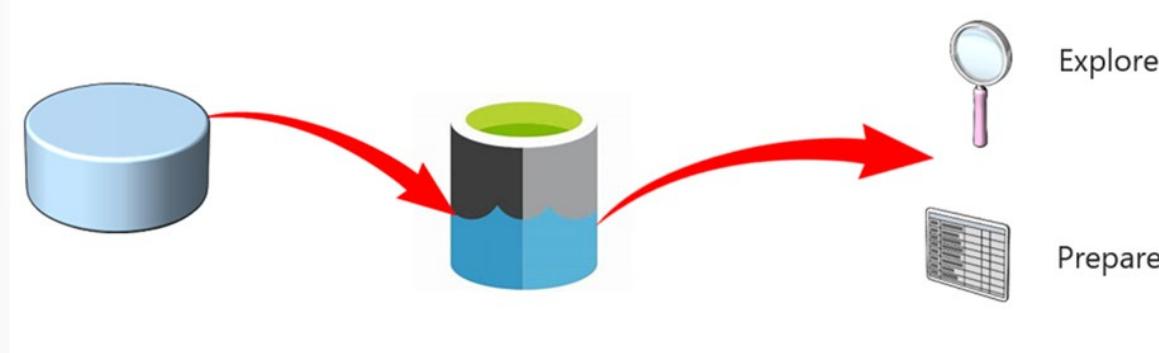
When it comes to analytics workloads, there are three major use cases:

Data exploration

Data exploration enables you to browse the data lake and get initial insights about the data, and is easily achievable with Azure Synapse Studio. Using this tool you can browse through the files, right click, and select TOP 100 rows just as you would do with a table in SQL Server. Once you are familiar with a data set, you can apply projections, filtering, grouping, and most of the operation over the data as if the data were in a regular SQL Server table.

Data transformation

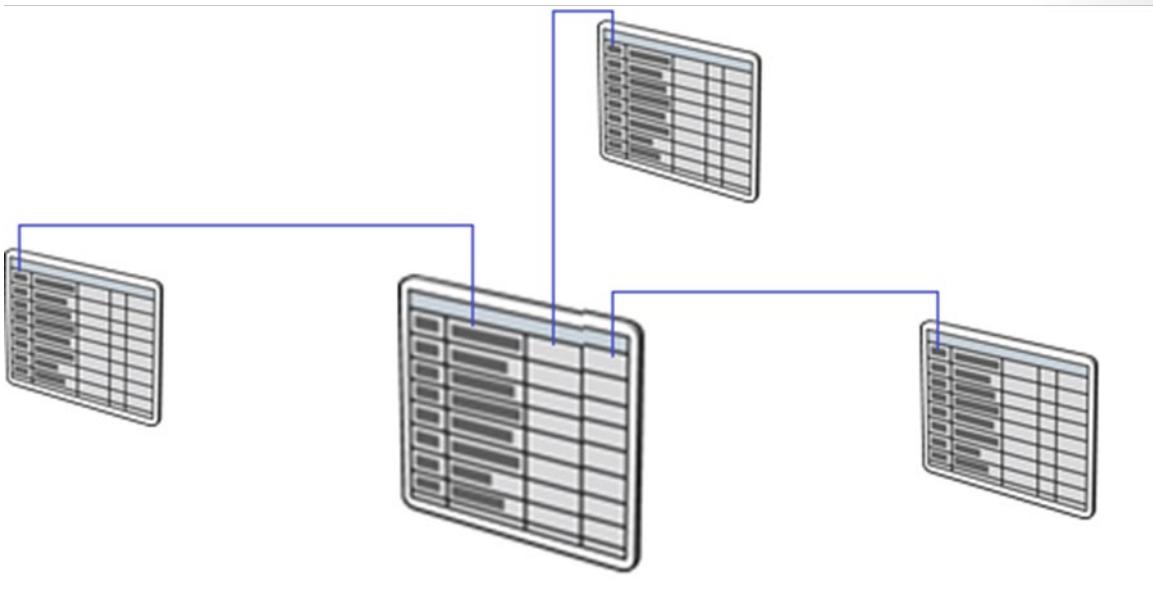
Azure Synapse Analytics provides great data transformations capabilities with Synapse Spark, however if you are skilled in SQL you might find data transformation easier to achieve. Serverless SQL pool enables you to execute a SELECT statement over the data in the lake and store the results back to the data lake in a specified format.



Logical data warehouse

Once you are familiar with the data you are interested in, you can start creating objects (such as VIEWS and External Tables) that provide you with a SQL metadata layer over the data in the lake. Once these objects are created, any tool that can connect to serverless SQL pool will see these objects as regular SQL Server objects. Clients don't even know that the underlying data is in the data lake. This makes serverless SQL pool a powerful lightweight layer between the data lake and a client. Given that serverless SQL pool heavily relies on SQL Server, a huge number of clients is supported. You can connect from BI tools like

Power BI or Azure Analysis Service, from integration tools like Synapse Pipelines or Azure Data Factory, and you can also programmatically connect using any of the popular languages like C# or Python.



Knowledge check

Question 1

Performing ad hoc data exploration in Azure Synapse Analytics uses which resource model?

- Dedicated SQL pools.
- Serverless SQL pools.
- Serverless Spark pools.

Question 2

What objects are used to provide a SQL metadata layer that creates a logical data warehouse in Azure Synapse Analytics?

- Views.
- Stored procedures.
- Functions.

Question 3

Which Azure storage service is the primary service used to query data from using Serverless SQL pools?

- Azure SQL Database.
- Azure Cosmos DB.
- Azure Data Lake.

Summary

Synapse workspace accelerates time to insight by enabling you to be productive and start producing insights without a need to move or copy the data from the data lake. There are two types of Synapse SQL pools: dedicated and serverless. Serverless SQL pool is charged per query based on amount of data processed by the query. You can use serverless SQL pool for three major use cases: Data exploration, Logical Data Warehouse, Data transformation.

In this lesson, you have learned how to:

- Identify Azure Synapse serverless SQL pools
- Describe when to use Azure Synapse serverless SQL pools
- Discuss Azure Synapse serverless SQL pools use cases

Query data in the lake using Azure Synapse serverless SQL pools

Introduction

Azure Synapse serverless SQL pool is tailored for querying the data in the lake. It supports querying CSV, JSON, and Parquet file formats directly. In this lesson you will learn how to craft queries to read a file (with specific format) and multiple files or folders. Additionally, you will learn how to extract specific data out of the files you are interested in. In this lesson, you will learn how you can query the different file types that can be stored in a data lake.

After the completion of this lesson, you will be able to:

- Query a CSV file using Azure Synapse serverless SQL pools
- Query a Parquet file using Azure Synapse serverless SQL pools
- Query a JSON file using Azure Synapse serverless SQL pools
- Query multiple files and folders using Azure Synapse serverless SQL pools
- Understand storage considerations when using Azure Synapse serverless SQL pool

Prerequisites

Before taking this lesson, it is recommended that the student is able to:

- Log into the Azure portal
- Explain the different components of Azure Synapse Analytics
- Use Azure Synapse Studio

Query a CSV file using Azure Synapse serverless SQL pools

CSV files are a common file format within many businesses, and you can query a single CSV file using serverless SQL pool. CSV files may have different formats:

- With and without a header row
- Comma and tab-delimited values
- Windows and Unix style line endings
- Non-quoted and quoted values, and escaping characters

All above variations will be covered below.

The **OPENROWSET** function enables you to read the content of CSV file by providing the URL to your file.

Read a csv file

The easiest way to see the content of your **CSV** file is to provide the **file URL** to the **OPENROWSET** function. You then include the csv **FORMAT**, and **2.0 PARSER_VERSION**. If the file is publicly available or

if your Azure AD identity can access this file, you should be able to see the content of the file using the query like the one shown in the following example:

```
select top 10 *
from openrowset(
    bulk 'https://pandemicdatalake.blob.core.windows.net/public/curated/
covid-19/ecdc_cases/latest/ecdc_cases.csv',
    format = 'csv',
    parser_version = '2.0',
    firstrow = 2 ) as rows
```

The option **firstrow** is used to skip the first row in the CSV file that represents the header in this case. Make sure that you can access this file. If your file is protected with a SAS key or custom identity, you would need to **setup server level credential for sql login**¹.

Data source usage

The previous example uses a full path to the file. As an alternative, you can create an external data source with the location that points to the root folder of the storage:

```
create external data source covid
with ( location = 'https://pandemicdatalake.blob.core.windows.net/public/
curated/covid-19/ecdc_cases' );
```

Once you create a data source, you can use that data source and the relative path to the file in OPEN-ROWSET function:

```
select top 10 *
from openrowset(
    bulk 'latest/ecdc_cases.csv',
    data_source = 'covid',
    format = 'csv',
    parser_version = '2.0',
    firstrow = 2
) as rows
```

If a data source is protected with SAS key or custom identity, you can configure **data source with database scoped credential**².

Explicitly specify a schema

The **OPENROWSET** function enables you to explicitly specify what columns you want to read from the file using WITH clause:

```
select top 10 *
from openrowset(
    bulk 'latest/ecdc_cases.csv',
    data_source = 'covid',
```

¹ <https://docs.microsoft.com/azure/synapse-analytics/sql/develop-storage-files-storage-access-control?tabs=shared-access-signature#server-scoped-credential>

² <https://docs.microsoft.com/azure/synapse-analytics/sql/develop-storage-files-storage-access-control?tabs=shared-access-signature#database-scoped-credential>

```
        format = 'csv',
        parser_version = '2.0',
        firstrow = 2
    ) with (
        date_rep date 1,
        cases int 5,
        geo_id varchar(6) 8
    ) as rows
```

The numbers after a data type in the WITH clause represent column location, known as a column index in the CSV file.

In the following sections, you can see how to query various types of CSV files. All the following examples require files that have specific row delimiter, column delimiter, escape character, etc. To be able to execute the upcoming examples, your first step is to create a database where the objects will be created. Then initialize the objects by executing the following **setup script**³ on that database. This setup script will create the data sources, database scoped credentials, and external file formats that are used in these samples.

Read windows style new line files

The following query shows how to read a CSV file without a header row, with a Windows-style new line, and comma-delimited columns, as shown in the following file preview example:

```
1 SI,Slovenia,2013,2095555CRLF
2 SI,Slovenia,2025,2093610CRLF
3 SI,Slovenia,2024,2096751CRLF
4 SI,Slovenia,2023,2099286CRLF
5 SI,Slovenia,2022,2101151CRLF
6 SI,Slovenia,2021,2102296CRLF
7 SI,Slovenia,2020,2102678CRLF
8 SI,Slovenia,2019,2102538CRLF
9 SI,Slovenia,2018,2102126CRLF
10 SI,Slovenia,2017,2101416CRLF
```

The code to achieve this is:

```
SELECT *
FROM OPENROWSET(
    BULK 'csv/population/population.csv',
    DATA_SOURCE = 'SqlOnDemandDemo',
    FORMAT = 'CSV', PARSER_VERSION = '2.0',
    FIELDTERMINATOR = ',',
    ROWTERMINATOR = '\n'
)
WITH (
    [country_code] VARCHAR (5) COLLATE Latin1_General_BIN2,
    [country_name] VARCHAR (100) COLLATE Latin1_General_BIN2,
    [year] smallint,
    [population] bigint
```

³ <https://github.com/Azure-Samples/Synapse/blob/master/SQL/Samples/LdwSample/SampleDB.sql>

```
) AS [r]
WHERE
    country_name = 'Luxembourg'
    AND year = 2017;
```

Read Unix-style new line files

The following query shows how to read a file without a header row, with a Unix-style new line, and comma-delimited columns as shown in the following file preview.

```
1 SI,Slovenia,2013,2095555LF
2 SI,Slovenia,2025,2093610LF
3 SI,Slovenia,2024,2096751LF
4 SI,Slovenia,2023,2099286LF
5 SI,Slovenia,2022,2101151LF
6 SI,Slovenia,2021,2102296LF
7 SI,Slovenia,2020,2102678LF
8 SI,Slovenia,2019,2102538LF
9 SI,Slovenia,2018,2102126LF
10 SI,Slovenia,2017,2101416LF
```

The code to achieve this is as follows. You can see in this example the different location of the file as compared to the other examples.

```
SELECT *
FROM OPENROWSET(
    BULK 'csv/population-unix/population.csv',
    DATA_SOURCE = 'SqlOnDemandDemo',
    FORMAT = 'CSV', PARSER_VERSION = '2.0',
    FIELDTERMINATOR = ',',
    ROWTERMINATOR = '0x0a' )
WITH (
    [country_code] VARCHAR (5) COLLATE Latin1_General_BIN2,
    [country_name] VARCHAR (100) COLLATE Latin1_General_BIN2,
    [year] smallint, [population] bigint
) AS [r]
WHERE
    country_name = 'Luxembourg'
    AND year = 2017;
```

Work with header rows in a file

The following query shows how to read a file with a header row, with a Unix-style new line, and comma-delimited columns, as shown in the following file preview.

```

1 country_code, country_name, year, populationLF
2 SI,Slovenia,2013,2095555LF
3 SI,Slovenia,2025,2093610LF
4 SI,Slovenia,2024,2096751LF
5 SI,Slovenia,2023,2099286LF
6 SI,Slovenia,2022,2101151LF
7 SI,Slovenia,2021,2102296LF
8 SI,Slovenia,2020,2102678LF
9 SI,Slovenia,2019,2102538LF
10 SI,Slovenia,2018,2102126LF

```

The code to achieve this is as follows where you use the **firstrow** variable to omit the header row.

```

SELECT *
FROM OPENROWSET(
    BULK 'csv/population-unix-hdr/population.csv',
    DATA_SOURCE = 'SqlOnDemandDemo',
    FORMAT = 'CSV', PARSER_VERSION = '2.0',
    FIELDTERMINATOR = ',',
    FIRSTROW = 2 )
WITH (
    [country_code] VARCHAR (5) COLLATE Latin1_General_BIN2,
    [country_name] VARCHAR (100) COLLATE Latin1_General_BIN2,
    [year] smallint, [population] bigint
) AS [r]
WHERE
    country_name = 'Luxembourg'
    AND year = 2017;

```

Work with custom quote character

The following query shows how to read a file with a header row, with a Unix-style new line, comma-delimited columns, and quoted value as shown in the following file preview.

```

1 country_code, country_name, year, populationLF
2 "SI","Slovenia",2013,2095555LF
3 "SI","Slovenia",2025,2093610LF
4 "SI","Slovenia",2024,2096751LF
5 "SI","Slovenia",2023,2099286LF
6 "SI","Slovenia",2022,2101151LF
7 "SI","Slovenia",2021,2102296LF
8 "SI","Slovenia",2020,2102678LF
9 "SI","Slovenia",2019,2102538LF
10 "SI","Slovenia",2018,2102126LF

```

The code to achieve this is as follows.

```

SELECT *
FROM OPENROWSET(
    BULK 'csv/population-unix-hdr-quoted/population.csv',
    DATA_SOURCE = 'SqlOnDemandDemo',
    FORMAT = 'CSV', PARSER_VERSION = '2.0',
    FIELDTERMINATOR = ',',
    QUOTED_IDENTIFIER = 'ALL')

```

```
FIELDTERMINATOR = ',',
ROWTERMINATOR = '0x0a',
FIRSTROW = 2,
FIELDQUOTE = ' " '
)
WITH (
    [country_code] VARCHAR (5) COLLATE Latin1_General_BIN2,
    [country_name] VARCHAR (100) COLLATE Latin1_General_BIN2,
    [year] smallint, [population] bigint
) AS [r]
WHERE
    country_name = 'Luxembourg'
    AND year = 2017;
```

NOTE:

This query would return the same results if you omitted the FIELDQUOTE parameter since the default value for FIELDQUOTE is a double-quote.

Work with escape characters

The following query shows how to read a file with a header row, with a Unix-style new line, comma-delimited columns, and an escape char used for the field delimiter (comma) within values, as shown in the following file preview.

```
1 country_code, country_name, year, populationLF
2 SI,Slov\,enia,2013,2095555LF
3 SI,Slovenia,2025,2093610LF
4 SI,Slovenia,2024,2096751LF
5 SI,Slovenia,2023,2099286LF
6 SI,Slovenia,2022,2101151LF
7 SI,Slovenia,2021,2102296LF
8 SI,Slovenia,2020,2102678LF
9 SI,Slovenia,2019,2102538LF
10 SI,Slovenia,2018,2102126LF
```

The code to achieve this is as follows.

```
SELECT *
FROM OPENROWSET(
    BULK 'csv/population-unix-hdr-escape/population.csv',
    DATA_SOURCE = 'SqlOnDemandDemo',
    FORMAT = 'CSV', PARSER_VERSION = '2.0',
    FIELDTERMINATOR = ',',
    ROWTERMINATOR = '0x0a',
    FIRSTROW = 2,
    ESCAPECHAR = ' \\ '
)
WITH (
    [country_code] VARCHAR (5) COLLATE Latin1_General_BIN2,
    [country_name] VARCHAR (100) COLLATE Latin1_General_BIN2,
    [year] smallint,
    [population] bigint
```

```
) AS [r]
WHERE
    country_name = 'Slovenia';
```

NOTE:

This query would fail if ESCAPECHAR is not specified since the comma in "Slov,enia" would be treated as field delimiter instead of part of the country/region name. "Slov,enia" would be treated as two columns. Therefore, the row would have one column more than the other rows, and one column more than you defined in the WITH clause.

Work with escape quoting characters

The following query shows how to read a file with a header row, with a Unix-style new line, comma-delimited columns, and an escaped double quote char within values as shown in the following file preview.

```
1  country_code,·country_name,·year,·populationLF
2  "SI","Slov""enia",2013,2095555LF
3  "SI","Slovenia",2025,2093610LF
4  "SI","Slovenia",2024,2096751LF
5  "SI","Slovenia",2023,2099286LF
6  "SI","Slovenia",2022,2101151LF
7  "SI","Slovenia",2021,2102296LF
8  "SI","Slovenia",2020,2102678LF
9  "SI","Slovenia",2019,2102538LF
10 "SI","Slovenia",2018,2102126LF
```

The code to achieve this is as follows.

```
SELECT *
FROM OPENROWSET(
    BULK 'csv/population-unix-hdr-escape-quoted/population.csv',
    DATA_SOURCE = 'SqlOnDemandDemo',
    FORMAT = 'CSV', PARSER_VERSION = '2.0',
    FIELDTERMINATOR = ',',
    ROWTERMINATOR = '0x0a',
    FIRSTROW = 2
)
WITH (
    [country_code] VARCHAR (5) COLLATE Latin1_General_BIN2,
    [country_name] VARCHAR (100) COLLATE Latin1_General_BIN2,
    [year] smallint,
    [population] bigint
) AS [r]
WHERE
    country_name = 'Slovenia';
```

NOTE:

The quoting character must be escaped with another quoting character. Quoting character can appear within column value only if value is encapsulated with quoting characters.

Work with tab-delimited files

The following query shows how to read a file with a header row, with a Unix-style new line, and tab-delimited columns, as shown in the following file preview.

```
1 country_code    country_name    year    populationLF
2 SI->Slovenia  ->2013      ->2095555LF
3 SI->Slovenia  ->2025      ->2093610LF
4 SI->Slovenia  ->2024      ->2096751LF
5 SI->Slovenia  ->2023      ->2099286LF
6 SI->Slovenia  ->2022      ->2101151LF
7 SI->Slovenia  ->2021      ->2102296LF
8 SI->Slovenia  ->2020      ->2102678LF
9 SI->Slovenia  ->2019      ->2102538LF
10 SI->Slovenia ->2018     ->2102126LF
```

The code to achieve this is as follows.

```
SELECT *
FROM OPENROWSET(
    BULK 'csv/population-unix-hdr-tsv/population.csv',
    DATA_SOURCE = 'SqlOnDemandDemo',
    FORMAT = 'CSV', PARSER_VERSION = '2.0',
    FIELDTERMINATOR = '\t',
    ROWTERMINATOR = '0x0a',
    FIRSTROW = 2
)
WITH (
    [country_code] VARCHAR (5) COLLATE Latin1_General_BIN2,
    [country_name] VARCHAR (100) COLLATE Latin1_General_BIN2,
    [year] smallint,
    [population] bigint
) AS [r]
WHERE
    country_name = 'Slovenia'
    AND year = 2017
```

Return a subset of columns from a file

So far, you've specified the CSV file schema using WITH and listing all columns. You can only specify columns you need in your query by using an ordinal number for each column needed. You'll also omit columns of no interest.

The following query returns the number of distinct country/region names in a file, specifying only the columns that are needed.

```
SELECT
    COUNT(DISTINCT country_name) AS countries
FROM OPENROWSET(
    BULK 'csv/population/population.csv',
    DATA_SOURCE = 'SqlOnDemandDemo',
    FORMAT = 'CSV', PARSER_VERSION = '2.0',
    FIELDTERMINATOR = ',', '
```

```

ROWTERMINATOR = '\n' )
WITH (
    --[country_code] VARCHAR (5),
    [country_name] VARCHAR (100) 2
    --[year] smallint,
    --[population] bigint
) AS [r]

```

NOTE:

Look at the WITH clause in the query below and note that there is "2" (without quotes) at the end of row where you define the [country_name] column. It means that the [country_name] column is the second column in the file. The query will ignore all columns in the file except the second one.

Query a Parquet file using Azure Synapse serverless SQL pools

You can also execute a query using serverless SQL pool that will read Parquet files. The **OPENROWSET** function enables you to read the content of a parquet file by providing the URL to your file.

Read parquet file

The easiest way to see the content of your PARQUET file is to provide the file **URL** to the **OPENROWSET** function and specify **parquet FORMAT**. If the file is publicly available, or if your Azure Active Directory identity can access this file, you should be able to see the content of the file using the query like the one shown in the following example:

```

select top 10 *
from openrowset(
    bulk 'https://pandemicdatalake.blob.core.windows.net/public/curated/
covid-19/ecdc_cases/latest/ecdc_cases.parquet',
    format = 'parquet') as rows

```

Make sure that you can access this file. If your file is protected with a SAS key or custom Azure identity, you would need to **setup server level credential for SQL login**⁴

Define a data source

The last example uses a full path to the file. As an alternative, you can create an external data source with the location that points to the root folder of the storage, and use that data source and the relative path to the file in the OPENROWSET function:

```

create external data source covid
with ( location = 'https://pandemicdatalake.blob.core.windows.net/public/
curated/covid-19/ecdc_cases' ) ; go
select top 10 *
from openrowset(
    bulk 'latest/ecdc_cases.parquet',
    data_source = 'covid',

```

⁴ <https://docs.microsoft.com/azure/synapse-analytics/sql/develop-storage-files-storage-access-control?tabs=shared-access-signature#server-scoped-credential>

```
        format = 'parquet'  
    ) as rows
```

If a data source is protected with a SAS key or custom identity, you can configure **data source with database scoped credential**⁵.

Explicitly specify a schema

OPENROWSET enables you to explicitly specify what columns you want to read from the file using the **WITH** clause:

```
select top 10 *  
from openrowset(  
    bulk 'latest/ecdc_cases.parquet',  
    data_source = 'covid',  
    format = 'parquet'  
) with (  
    date_rep date,  
    cases int,  
    geo_id varchar(6)  
) as rows
```

In the following sections you can see how to query various types of PARQUET files. All the following examples require specific file layouts. To be able to execute the upcoming examples, your first step is to **create a database** where the objects will be created. Then initialize the objects by executing the **setup script**⁶ on that database. This setup script will create the data sources, database scoped credentials, and external file formats that are used in these samples.

The NYC Yellow Taxi <https://azure.microsoft.com/services/open-datasets/catalog/nyc-taxi-limousine-commission-yellow-taxi-trip-records/> dataset is used in this sample.

You can query Parquet files the same way you read CSV files. The only difference is that the FILEFORMAT parameter should be set to PARQUET. Examples in this article show the specifics of reading Parquet files.

Query specific column of parquet files

You can specify only the columns of interest when you query Parquet files using the **with** clause as shown in the following code segment.

```
SELECT  
    YEAR(tpepPickupDateTime),  
    passengerCount,  
    COUNT(*) AS cnt  
FROM  
OPENROWSET(  
    BULK 'puYear=2018/puMonth=/*/*.snappy.parquet',  
    DATA_SOURCE = 'YellowTaxi',  
    FORMAT='PARQUET'  
) WITH (
```

⁵ <https://docs.microsoft.com/azure/synapse-analytics/sql/develop-storage-files-storage-access-control?tabs=shared-access-signature#database-scoped-credential>

⁶ <https://github.com/Azure-Samples/Synapse/blob/master/SQL/Samples/LdwSample/SampleDB.sql>

```

        tpepPickupDateTime DATETIME2,
        passengerCount INT
    ) AS nyc
GROUP BY
    passengerCount,
    YEAR(tpepPickupDateTime)
ORDER BY
    YEAR(tpepPickupDateTime),
    passengerCount;

```

Although you don't need to use the OPENROWSET WITH clause when reading Parquet files. Column names and data types are automatically read from Parquet files.

The sample below shows the automatic schema inference capabilities for Parquet files. It returns the number of rows in 2018 without specifying a schema.

```

SELECT TOP 10 *
FROM
OPENROWSET(
    BULK 'puYear=2018/puMonth=/*/*.snappy.parquet',
    DATA_SOURCE = 'YellowTaxi',
    FORMAT='PARQUET'
) AS nyc

```

Query partitioned data

The data set provided in the NYC Yellow Taxi dataset is partitioned (or divided) into separate subfolders. You can target specific partitions using the **filepath** function. This example shows fare amounts by year, month, and payment_type for the first three months of 2017.

NOTE:

The serverless SQL pool query is compatible with Hive/Hadoop partitioning scheme.

```

SELECT
    YEAR(tpepPickupDateTime),
    passengerCount,
    COUNT(*) AS cnt
FROM
OPENROWSET(
    BULK 'puYear=*/puMonth=/*/*.snappy.parquet',
    DATA_SOURCE = 'YellowTaxi',
    FORMAT='PARQUET' ) nyc
WHERE
    nyc.filepath(1) = 2017
    AND nyc.filepath(2) IN (1, 2, 3)
    AND tpepPickupDateTime BETWEEN CAST('1/1/2017' AS datetime) AND
    CAST('3/31/2017' AS datetime)
GROUP BY
    passengerCount,
    YEAR(tpepPickupDateTime)
ORDER BY

```

```
YEAR(tpepPickupDateTime),  
passengerCount;
```

Data type mapping

Parquet files contain type descriptions for every column. The following table describes how Parquet types are mapped to SQL data types.

Parquet type	Parquet logical type (annotation)	SQL data type
BOOLEAN		bit
BINARY / BYTE_ARRAY		varbinary
DOUBLE		float
FLOAT		real
INT32		int
INT64		bigint
INT96		datetime2
FIXED_LEN_BYTE_ARRAY		binary
BINARY	UTF8	varchar *(UTF8 collation)
BINARY	STRING	varchar *(UTF8 collation)
BINARY	ENUM	varchar *(UTF8 collation)
BINARY	UUID	uniqueidentifier
BINARY	DECIMAL	decimal
BINARY	JSON	varchar(max) *(UTF8 collation)
BINARY	BSON	varbinary(max)
FIXED_LEN_BYTE_ARRAY	DECIMAL	decimal
BYTE_ARRAY	INTERVAL	varchar(max), serialized into standardized format
INT32	INT(8, true)	smallint
INT32	INT(16, true)	smallint
INT32	INT(32, true)	int
INT32	INT(8, false)	tinyint
INT32	INT(16, false)	int
INT32	INT(32, false)	bigint
INT32	DATE	date
INT32	DECIMAL	decimal
INT32	TIME (MILLIS)	time
INT64	INT(64, true)	bigint
INT64	INT(64, false)	decimal(20,0)
INT64	DECIMAL	decimal
INT64	TIME (MICROS / NANOS)	time
INT64	TIMESTAMP (MILLIS / MICROS / NANOS)	datetime2
Complex type	LIST	varchar(max), serialized into JSON

Parquet type	Parquet logical type (annotation)	SQL data type
Complex type	MAP	varchar(max), serialized into JSON

Query a JSON file using Azure Synapse serverless SQL pools

You can also query JSON files using serverless SQL pools. The query's objective is to read the following type of JSON files using **OPENROWSET**.

- Standard JSON files where multiple JSON documents are stored as a JSON array.
- Line-delimited JSON files, where JSON documents are separated with new-line character. Common extensions for these types of files are jsonl, ldjson, and ndjson.

Read JSON documents

The easiest way to see the content of your JSON file is to provide the file URL to the OPENROWSET function, specify **csv FORMAT**, and set the values of **0x0b** for the **fieldterminator** and **fieldquote** variables. If you need to read line-delimited JSON files, then this is enough. If you have classic JSON file, you would need to set values **0x0b** for **rowterminator**. The OPENROWSET function will parse JSON and return every document in the following format:

```
{"date_rep": "2020-07-24", "day": 24, "month": 7, "year": 2020, "cases": 3, "deaths": 0, "geo_id": "AF"}  
{"date_rep": "2020-07-25", "day": 25, "month": 7, "year": 2020, "cases": 7, "deaths": 0, "geo_id": "AF"}  
{"date_rep": "2020-07-26", "day": 26, "month": 7, "year": 2020, "cases": 4, "deaths": 0, "geo_id": "AF"}  
{"date_rep": "2020-07-27", "day": 27, "month": 7, "year": 2020, "cases": 8, "deaths": 0, "geo_id": "AF"}
```

If the file is publicly available, or if your Azure Active Directory identity can access this file, you should see the content of the file using the query like the one shown in the following examples.

Read JSON files

The following two sample queries read JSON and line-delimited JSON files (.JSONL). The first query is reading line-delimited JSON file and you can find example below.

```
select top 10 *
from
    openrowset(
        bulk 'https://pandemicdatalake.blob.core.windows.net/public/curated/covid-19/ecdc_cases/latest/ecdc_cases.jsonl',
        format = 'csv',
        fieldterminator = '0x0b',
        fieldquote = '0x0b'
    ) with (doc nvarchar(max)) as rows
```

The second query is reading standard JSON file, and you can find that example below.

```
select top 10 *
from
    openrowset(
        bulk 'https://pandemicdatalake.blob.core.windows.net/public/curated/covid-19/ecdc_cases/latest/ecdc_cases.json',
        format = 'csv',
        fieldterminator = '0x0b',
        fieldquote = '0x0b',
        rowterminator = '0x0b' --> You need to override rowterminator to
read classic JSON
    ) with (doc nvarchar(max)) as rows
```

This second example is reading the classic JSON file:

```
select top 10 *
from
    openrowset(
        bulk 'https://pandemicdatalake.blob.core.windows.net/public/curated/covid-19/ecdc_cases/latest/ecdc_cases.json',
        format = 'csv',
        fieldterminator = '0x0b',
        fieldquote = '0x0b',
        rowterminator = '0x0b' --> You need to override rowterminator to
read classic JSON
    ) with (doc nvarchar(max)) as rows
```

The JSON document in the preceding sample query includes an array of objects. The query returns each object as a separate row in the result set. Make sure that you can access this file. If your file is protected with a SAS key or custom identity, you would need to set up **server level credential for sql login**⁷

Define a data source

The previous example uses a full path to the file. As an alternative, you can create an external data source with the location that points to the root folder of the storage, and use that data source and the relative path to the file in the OPENROWSET function:

```
create external data source covid
with (location = 'https://pandemicdatalake.blob.core.windows.net/public/
curated/covid-19/ecdc_cases');
go
select top 10 *
from
    openrowset(
        bulk 'latest/ecdc_cases.json',
        data_source = 'covid',
        format = 'csv',
        fieldterminator = '0x0b',
        fieldquote = '0x0b'
```

⁷ <https://docs.microsoft.com/azure/synapse-analytics/sql/develop-storage-files-storage-access-control?tabs=shared-access-signature#server-scoped-credential>

```

) with (doc nvarchar(max)) as rows
go
select top 10 *
from
openrowset(
    bulk 'latest/ecdc_cases.json',
    data_source = 'covid',
    format = 'csv',
    fieldterminator = '0x0b',
    fieldquote = '0x0b',
    rowterminator = '0x0b' --> You need to override rowterminator to
read classic JSON
) with (doc nvarchar(max)) as rows

```

If a data source is protected with a SAS key or custom identity you can configure **data source with database scoped credential**⁸

In the following sections, you can see how to query various types of JSON files.

Parse JSON documents

The queries in the previous examples return every JSON document as a single string in a separate row of the result set. You can use functions JSON_VALUE and OPENJSON to parse the values in JSON documents and return them as relational values, as it's shown in the following example:

PARSE JSON DOCUMENTS

date_rep	cases	geo_id
2020-07-24	3	AF
2020-07-25	7	AF
2020-07-26	4	AF
2020-07-27	8	AF

Which is a json file containing the following structure:

```
{
    "date_rep": "2020-07-24",
    "day": 24,
    "month": 7,
    "year": 2020,
    "cases": 13,
    "deaths": 0,
    "countries_and_territories": "Afghanistan",
    "geo_id": "AF",
    "country_territory_code": "AFG",
    "continent_exp": "Asia",
    "load_date": "2020-07-25 00:05:14",
    "iso_country": "AF"
}
```

⁸ <https://docs.microsoft.com/azure/synapse-analytics/sql/develop-storage-files-storage-access-control?tabs=shared-access-signature#database-scoped-credential>

NOTE:

If these documents are stored as line-delimited JSON, you need to set **FIELDTERMINATOR** and **FIELDQUOTE** to **0x0b**. If you have standard JSON format you need to set **ROWTERMINATOR** to **0x0b**.

Query JSON files using JSON_VALUE

The following query below shows you how to use **JSON_VALUE**⁹ to retrieve scalar values (title, publisher) from JSON documents:

```
select
    JSON_VALUE(doc, '$.date_rep') AS date_reported,
    JSON_VALUE(doc, '$.countries_and_territories') AS country,
    JSON_VALUE(doc, '$.cases') as cases,
    doc
from
    openrowset(
        bulk 'latest/ecdc_cases.jsonl',
        data_source = 'covid',
        format = 'csv',
        fieldterminator ='0x0b',
        fieldquote = '0x0b'
    ) with (doc nvarchar(max)) as rows
order by JSON_VALUE(doc, '$.geo_id') desc
```

Query JSON files using OPENJSON

The following query uses **OPENJSON**¹⁰. It will retrieve COVID statistics reported in Serbia:

```
select *
from
    openrowset(
        bulk 'latest/ecdc_cases.jsonl',
        data_source = 'covid',
        format = 'csv',
        fieldterminator ='0x0b',
        fieldquote = '0x0b'
    ) with (doc nvarchar(max)) as rows
cross apply openjson (doc)
    with (
        date_rep datetime2,
        cases int,
        fatal int '$.deaths',
        country varchar(100) '$.countries_and_territories'
    )
where country = 'Serbia'
order by country, date_rep desc;
```

⁹ <https://docs.microsoft.com/sql/t-sql/functions/json-value-transact-sql?toc=/azure/synapse-analytics/toc.json&bc=/azure/synapse-analytics/breadcrumb/toc.json&view=azure-sqldw-latest>

¹⁰ <https://docs.microsoft.com/sql/t-sql/functions/openjson-transact-sql?toc=/azure/synapse-analytics/toc.json&bc=/azure/synapse-analytics/breadcrumb/toc.json&view=azure-sqldw-latest>

Query multiple files and folders using Azure Synapse serverless SQL pools

Serverless SQL pools supports reading multiple files or folders by using wildcards, which are similar to **wildcards used in Windows OS¹¹**. However, greater flexibility is present since multiple wildcards are allowed.

Your first step is to create a database where you will execute the queries. Then initialize the objects by executing the following **setup script¹²** on that database. This setup script will create the data sources, database scoped credentials, and external file formats that are used in these samples.

You will use the folder csv/taxi to follow the sample queries. It contains NYC Taxi - Yellow Taxi Trip Records data from July 2016 to June 2018. Files in csv/taxi folder are named after years and month using the following pattern: yellow_tripdata_-.csv

Read all files in folder

The example below reads all NYC Yellow Taxi data files from the csv/taxi folder and returns the total number of passengers and rides per year. It also shows usage of aggregate functions.

```
SELECT
    YEAR(pickup_datetime) AS [year],
    SUM(passenger_count) AS passengers_total,
    COUNT(*) AS [rides_total]
FROM
    OPENROWSET(
        BULK 'csv/taxi/*.csv',
        DATA_SOURCE = 'sqlondemanddemo',
        FORMAT = 'CSV',
        PARSER_VERSION = '2.0',
        FIRSTROW = 2
    ) WITH (
        pickup_datetime DATETIME2 2,
        passenger_count INT 4
    ) AS nyc
GROUP BY
    YEAR(pickup_datetime)
ORDER BY
    YEAR(pickup_datetime);
```

NOTE:

All files accessed with the single OPENROWSET must have the same structure (i.e., number of columns and their data types).

Read subset of files in folder

The example below reads the 2017 NYC Yellow Taxi data files from the csv/taxi folder using a wildcard and returns the total fare amount per payment type.

¹¹ <https://docs.microsoft.com/previous-versions/windows/desktop/indexsrv/ms-dos-and-windows-wildcard-characters>

¹² <https://github.com/Azure-Samples/Synapse/blob/master/SQL/Samples/LdwSample/SampleDB.sql>

```
SELECT
    payment_type,
    SUM(fare_amount) AS fare_total
FROM
    OPENROWSET(
        BULK 'csv/taxi/yellow_tripdata_2017-* .csv',
        DATA_SOURCE = 'sqlondemanddemo',
        FORMAT = 'CSV',
        PARSER_VERSION = '2.0',
        FIRSTROW = 2
    ) WITH (
        payment_type INT 10,
        fare_amount FLOAT 11
    ) AS nyc
GROUP BY
    payment_type
ORDER BY
    payment_type;
```

NOTE:

All files accessed with the single OPENROWSET must have the same structure (i.e., number of columns and their data types).

Read folders

The path that you provide to OPENROWSET can also be a path to a folder. The following sections include these query types.

Read all files from a specific folder

You can read all the files in a folder using the file level wildcard as shown in Read all files in folder. But there is a way to query a folder and consume all files within that folder.

If the path provided in the OPENROWSET points to a folder, all files in that folder will be used as a source for your query. The following query will read all files in the csv/taxi folder.

NOTE:

Note the existence of the / at the end of the path in the query below. It denotes a folder. If the / is omitted, the query will target a file named taxi instead.

```
SELECT
    YEAR(pickup_datetime) as [year],
    SUM(passenger_count) AS passengers_total,
    COUNT(*) AS [rides_total]
FROM OPENROWSET(
    BULK 'csv/taxi/',
    DATA_SOURCE = 'sqlondemanddemo',
    FORMAT = 'CSV',
    PARSER_VERSION = '2.0',
    FIRSTROW = 2
) WITH (
    vendor_id VARCHAR(100) COLLATE Latin1_General_BIN2,
```

```

pickup_datetime DATETIME2,
dropoff_datetime DATETIME2,
passenger_count INT,
trip_distance FLOAT,
rate_code INT,
store_and_fwd_flag VARCHAR(100) COLLATE Latin1_General_BIN2,
pickup_location_id INT,
dropoff_location_id INT,
payment_type INT,
fare_amount FLOAT,
extra FLOAT,
mta_tax FLOAT,
tip_amount FLOAT,
tolls_amount FLOAT,
improvement_surcharge FLOAT,
total_amount FLOAT
) AS nyc
GROUP BY
    YEAR(pickup_datetime)
ORDER BY
    YEAR(pickup_datetime);

```

NOTE:

All files accessed with the single OPENROWSET must have the same structure (i.e., number of columns and their data types).

Read all files from multiple folders

It's possible to read files from multiple folders by using a wildcard. The following query will read all files from all folders located in the csv folder that have names starting with t and ending with i.

```

SELECT
    YEAR(pickup_datetime) as [year],
    SUM(passenger_count) AS passengers_total,
    COUNT(*) AS [rides_total]
FROM OPENROWSET(
    BULK 'csv/t*i/',
    DATA_SOURCE = 'sqlondemanddemo',
    FORMAT = 'CSV', PARSER_VERSION = '2.0',
    FIRSTROW = 2
)
WITH (
    vendor_id VARCHAR(100) COLLATE Latin1_General_BIN2,
    pickup_datetime DATETIME2,
    dropoff_datetime DATETIME2,
    passenger_count INT,
    trip_distance FLOAT,
    rate_code INT,
    store_and_fwd_flag VARCHAR(100) COLLATE Latin1_General_BIN2,
    pickup_location_id INT,
    dropoff_location_id INT,
)

```

```
payment_type INT,
fare_amount FLOAT,
extra FLOAT,
mta_tax FLOAT,
tip_amount FLOAT,
tolls_amount FLOAT,
improvement_surcharge FLOAT,
total_amount FLOAT
) AS nyc
GROUP BY
YEAR(pickup_datetime)
ORDER BY
YEAR(pickup_datetime);
```

NOTE:

All files accessed with the single OPENROWSET must have the same structure (i.e., number of columns and their data types).

Since you have only one folder that matches the criteria, the query result is the same as Read all files in folder.

Use multiple wildcards

You can use multiple wildcards on different path levels. For example, you can enrich previous query to read files with 2017 data only, from all folders which names start with t and end with i.

NOTE:

The existence of the / at the end of the path in the query below. It denotes a folder. If the / is omitted, the query will target files named t*i instead. There is a maximum limit of 10 wildcards per query.

```
SELECT
YEAR(pickup_datetime) as [year],
SUM(passenger_count) AS passengers_total,
COUNT(*) AS [rides_total]
FROM OPENROWSET(
    BULK 'csv/t*i/yellow_tripdata_2017-*.csv',
    DATA_SOURCE = 'sqlondemanddemo',
    FORMAT = 'CSV', PARSER_VERSION = '2.0',
    FIRSTROW = 2
)
WITH (
    vendor_id VARCHAR(100) COLLATE Latin1_General_BIN2,
    pickup_datetime DATETIME2,
    dropoff_datetime DATETIME2,
    passenger_count INT,
    trip_distance FLOAT,
    rate_code INT,
    store_and_fwd_flag VARCHAR(100) COLLATE Latin1_General_BIN2,
    pickup_location_id INT,
    dropoff_location_id INT,
    payment_type INT,
    fare_amount FLOAT,
    extra FLOAT,
```

```
        mta_tax FLOAT,
        tip_amount FLOAT,
        tolls_amount FLOAT,
        improvement_surcharge FLOAT,
        total_amount FLOAT
    ) AS nyc
GROUP BY
    YEAR(pickup_datetime)
ORDER BY
    YEAR(pickup_datetime);
```

NOTE:

All files accessed with the single OPENROWSET must have the same structure (i.e., number of columns and their data types).

Storage considerations when using Azure Synapse serverless SQL pools

As enterprises deploy performance sensitive cloud-native applications, it's important to have options for cost-effective data storage at different performance levels.

Azure block blob storage offers two different performance tiers:

- **Premium:** optimized for high transaction rates and single-digit consistent storage latency
- **Standard:** optimized for high capacity and high throughput

Premium performance block blob storage makes data available via high-performance hardware. Data is stored on solid-state drives (SSDs) which are optimized for low latency. SSDs provide higher throughput compared to traditional hard drives.

Premium performance storage is ideal for workloads that require fast and consistent response times. It's best for workloads that perform many small transactions. Example workloads include:

Interactive workloads.

These workloads require instant updates and user feedback, such as e-commerce and mapping applications. For example, in an e-commerce application, less frequently viewed items are likely not cached. However, they must be instantly displayed to the customer on demand.

Analytics.

In an IoT scenario, lots of smaller write operations might be pushed to the cloud every second. Large amounts of data might be taken in, aggregated for analysis purposes, and then deleted almost immediately. The high ingestion capabilities of premium block blob storage make it efficient for this type of workload.

Artificial intelligence/machine learning (AI/ML).

AI/ML deals with the consumption and processing of different data types like visuals, speech, and text. This high-performance computing type of workload deals with large amounts of data that requires rapid response and efficient ingestion times for data analysis.

Data transformation.

Processes that require constant editing, modification, and conversion of data require instant updates. For accurate data representation, consumers of this data must see these changes reflected immediately.

Azure Data Lake Storage Gen2 provides file system performance at object storage scale and prices using **hierarchical namespace** feature. This allows the collection of objects/files within an account to be organized into a hierarchy of directories and nested subdirectories in the same way that the file system on your computer is organized. With a hierarchical namespace enabled, a storage account becomes capable of providing the scalability and cost-effectiveness of object storage, with file system semantics that are familiar to analytics engines and frameworks.

The following benefits are associated with file systems that implement a hierarchical namespace over blob data:

- **Atomic directory manipulation:** Object stores approximate a directory hierarchy by adopting a convention of embedding slashes (/) in the object name to denote path segments. While this convention works for organizing objects, the convention provides no assistance for actions like moving, renaming, or deleting directories. Without real directories, applications must process potentially millions of individual blobs to achieve directory-level tasks. By contrast, a hierarchical namespace processes these tasks by updating a single entry (the parent directory).

This dramatic optimization is especially significant for many big data analytics frameworks. Tools like Hive, Spark, etc. often write output to temporary locations and then rename the location at the conclusion of the job. Without a hierarchical namespace, this rename can often take longer than the analytics process itself. Lower job latency equals lower total cost of ownership (TCO) for analytics workloads.

- **Familiar Interface Style:** File systems are well understood by developers and users alike. There is no need to learn a new storage paradigm when you move to the cloud as the file system interface exposed by Data Lake Storage Gen2 is the same paradigm used by computers, large and small.

One of the reasons that object stores haven't historically supported a hierarchical namespace is that a hierarchical namespace limits scale. However, the Data Lake Storage Gen2 hierarchical namespace scales linearly and does not degrade either the data capacity or performance.

Practically speaking, if you are looking for the best performance for serverless SQL pool you should go for Azure Data Lake Storage Gen2 premium tier. Please note that this offer has the largest cost.

Knowledge check

Question 1

What function is used to read the data in files stored in a data lake?

- FORMAT.
- ROWSET.
- OPENROWSET.

Question 2

What value should be set in the fieldterminator and fieldquote variables to read JSON files?

- 0b0x.
- 00xb.
- 0x0b.

Question 3

What character in file path can be used to select all the file/folders that match rest of the path?

- &.
- *.
- /.

Summary

Serverless SQL pools enable you to easily query files in data lake. You can query various file formats CSV, JSON, Parquet. Using familiar T-SQL syntax and few new capabilities you can easily go from raw files in data lake to useful insights for your business.

In this lesson, you have learned how to:

- Query a CSV file using Azure Synapse serverless SQL pools
- Query a Parquet file using Azure Synapse serverless SQL pools
- Query a JSON file using Azure Synapse serverless SQL pools
- Query multiple files and folders using Azure Synapse serverless SQL pools
- Understand storage considerations when using Azure Synapse serverless SQL pool

Create metadata objects in Azure Synapse serverless SQL pools

Introduction

In this lesson, you will learn how you can create objects to help you query data or optimize your existing data transformation pipeline through Azure Synapse serverless SQL pools.

After the completion of this lesson, you will be able to:

- Create databases in Azure Synapse serverless SQL pools
- Create and manage credentials in Azure Synapse serverless SQL pools
- Create external data sources in Azure Synapse serverless SQL pools
- Create external tables in Azure Synapse serverless SQL pools
- Create views in Azure Synapse serverless SQL pools

Prerequisites

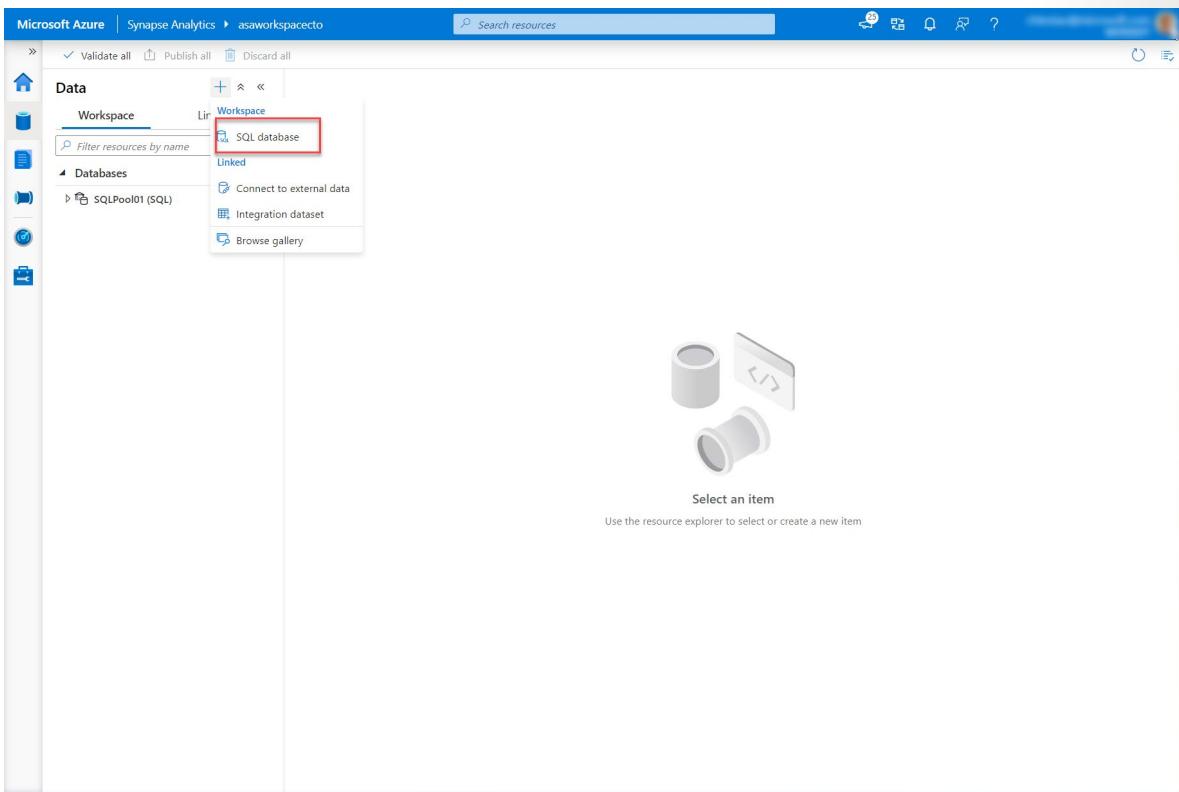
Before taking this lesson, it is recommended that the student is able to:

- Log into the Azure portal
- Explain the different components of Azure Synapse Analytics
- Use Azure Synapse Studio

Create databases in Azure Synapse serverless SQL pools

Once you are familiar with your data, you can define the metadata objects inside of serverless SQL pool so the clients connecting to it can discover them and query the data in the lake. The first metadata object you will create is a database.

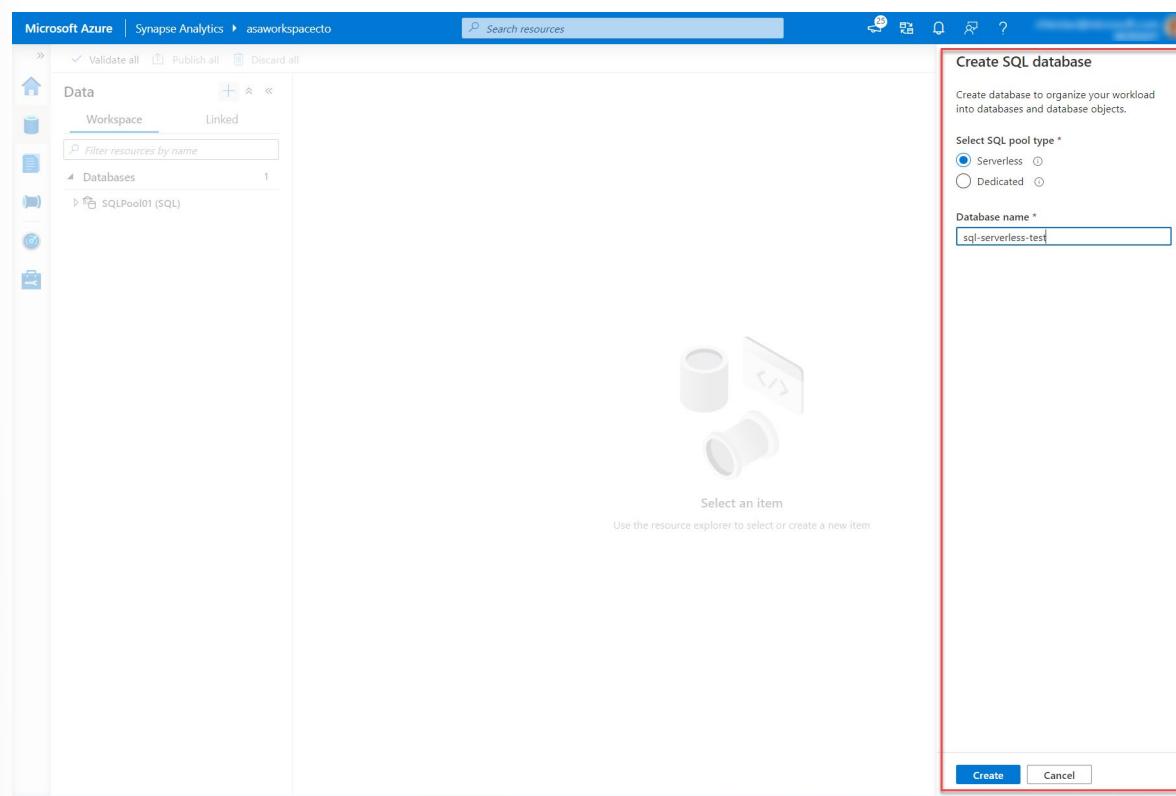
To create a new database as an Azure Synapse serverless SQL pool, go to the **Data** hub, click + button and select **SQL database**.



Choose one of the following two pool type options:

- Serverless
- Dedicated

Select **Serverless** if it isn't already selected, type in a **database name** and press the button **Create**.



In addition to using the Azure Synapse Studio user interface to create a database, you can execute the following Transact-SQL statement in the serverless SQL pool:

```
CREATE DATABASE [YourDatabaseName]
```

Create and manage credentials in Azure Synapse serverless SQL pools

Azure Synapse serverless SQL pool accesses the storage to read the files using credentials. There are 3 types of credentials that are supported:

- Azure Active Directory pass-through
- Managed Identity
- Shared access signature (SAS).

Azure Active Directory pass-through, is the default credential used for Azure Active Directory users. If you login with an Azure Active Directory identity, and you don't explicitly specify the credential, the identity of the logged in user will be used. To explicitly specify an identity of a logged in user, you need to create a database scoped credential as follows:

```
CREATE DATABASE SCOPED CREDENTIAL [sqlondemand]
WITH IDENTITY='User Identity'
```

Managed identities is an Azure Active Directory identity that represents the the Azure Synapse workspace in your Azure Active Directory tenant. You can grant access to services to this identity explicitly. To instruct the serverless SQL pool to use a managed identity, create a credential as follows.

```
CREATE DATABASE SCOPED CREDENTIAL [sqlondemand]
WITH IDENTITY='Managed Identity'
```

A Shared access signature (SAS) is a storage feature, that enables you to give a time bound permissions at a storage account file system or directory level . To use a SAS credential in serverless SQL pool, , create a credential as follows.

```
CREATE DATABASE SCOPED CREDENTIAL [sqlondemand]
WITH IDENTITY='SHARED ACCESS SIGNATURE',
SECRET = 'sv=2018-03-28&ss=bf&srt=sco&sp=rl&st=2019-10-14T12%3A10%3A25Z&se=
2061-12-31T12%3A10%3A00Z&sig=K1SU2ullCscyTS0An0nozEpo4tO5JAgGBvw%2FJX2lgu-
w%3D'
```

Whichever credential you choose to use, all credentials can be passed to an external data source, defining what authentication method should be used against the storage.

Create external data sources in Azure Synapse serverless SQL pools

An external data source is used to define the location of the data and the credential that should be used to access it. You can create external data source to a public storage account as follows:

```
CREATE EXTERNAL DATA SOURCE YellowTaxi
WITH ( LOCATION = 'https://azureopendatastorage.blob.core.windows.net/
nyctlc/yellow/' )
```

If the storage account is protected, you must specify which credentials should be used like this:

```
CREATE EXTERNAL DATA SOURCE SqlOnDemandDemo WITH (
    LOCATION = 'https://sqlondemandstorage.blob.core.windows.net',
    CREDENTIAL = sqlondemand
);
```

External data source are typically used in the **OPENROWSET** function or as part of the external table definition.

Create external tables in Azure Synapse serverless SQL pools

External tables are useful when you want to control access to external data in serverless SQL pools and is commonly used in PolyBase activities. If you want to use tools, such as Power BI, in conjunction with serverless SQL pools, then external tables are needed. External tables can access two types of storage:

- Public storage where users access public storage files.
- Protected storage where users access storage files using SAS credential, Azure AD identity, or the Managed Identity of an Azure Synapse workspace.



<https://channel9.msdn.com/Shows/Learn-Azure/Introduction-to-Polybase/player?format=ny>

Prerequisites

Your first step is to create a database where the tables will be created. Then initialize the objects by executing the following **setup script**¹³ on that database. This setup script will create the following objects that are used in this sample:

- **https://sqlondemandstorage.blob.core.windows.net** Azure storage account.
- DATABASE SCOPED CREDENTIAL sqlondemand that enables access to SAS-protected

```
CREATE DATABASE SCOPED CREDENTIAL [sqlondemand]
WITH IDENTITY='SHARED ACCESS SIGNATURE',
SECRET = 'sv=2018-03-28&ss=bf&srt=sco&sp=rl&st=2019-10-14T12%3A10%3A25Z&se=2061-12-3
1T12%3A10%3A00Z&sig=KISU2ullCscyTS0An0nozEpo4tO5JAgGBvw%2FJX2Iguw%3D'
```

- EXTERNAL DATA SOURCE sqlondemanddemo that references demo storage account protected with SAS key, and EXTERNAL DATA SOURCE YellowTaxi that references publicly available Azure storage account on location <https://azureopendatastorage.blob.core.windows.net/nyctlc/yellow/>

```
CREATE EXTERNAL DATA SOURCE SqlOnDemandDemo WITH (
    LOCATION = 'https://sqlondemandstorage.blob.core.windows.net',
    CREDENTIAL = sqlondemand
);
GO
CREATE EXTERNAL DATA SOURCE YellowTaxi
WITH ( LOCATION = 'https://azureopendatastorage.blob.core.windows.net/nyctlc/yellow/' )
```

- Next you will create a file format named QuotedCSVWithHeaderFormat and FORMAT_TYPE of parquet that defines two file types. CSV and parquet.

```
CREATE EXTERNAL FILE FORMAT QuotedCsvWithHeaderFormat
WITH (
    FORMAT_TYPE = DELIMITEDTEXT,
    FORMAT_OPTIONS ( FIELD_TERMINATOR = ',', STRING_DELIMITER = "'", FIRST_ROW = 2 )
);
GO
CREATE EXTERNAL FILE FORMAT ParquetFormat WITH ( FORMAT_TYPE = PARQUET );
```

¹³ <https://github.com/Azure-Samples/Synapse/blob/master/SQL/Samples/LdwSample/SampleDB.sql>

Create an external table on protected data

With the database scoped credential, external data source, and external file format defined, you can create external tables that access data on an Azure storage account that allows access to users with some Azure AD identity or SAS key. You can create external tables the same way you create regular SQL Server external tables. The following query creates an external table that reads the population.csv file from the SynapseSQL demo Azure storage account that is referenced using sqlondemanddemo data source and is protected with the database scoped credential called sqlondemand. The data source and database scoped credential are created in the **setup script**¹⁴.

NOTE:

Change the first line in the query, i.e. [mydbname], so you're using the database you created.

```
USE [mydbname];
GO
CREATE EXTERNAL TABLE populationExternalTable
(
    [country_code] VARCHAR (5) COLLATE Latin1_General_BIN2,
    [country_name] VARCHAR (100) COLLATE Latin1_General_BIN2,
    [year] smallint,
    [population] bigint
)
WITH (
    LOCATION = 'csv/population/population.csv',
    DATA_SOURCE = sqlondemanddemo,
    FILE_FORMAT = QuotedCSVWithHeaderFormat
);
```

Create an external table on public data

You can create external tables that read data from the files placed on publicly available Azure storage. This setup script will create a public external data source with a Parquet file format definition that is used in the following query:

```
CREATE EXTERNAL TABLE Taxi (
    vendor_id VARCHAR(100) COLLATE Latin1_General_BIN2,
    pickup_datetime DATETIME2,
    dropoff_datetime DATETIME2,
    passenger_count INT,
    trip_distance FLOAT,
    fare_amount FLOAT,
    tip_amount FLOAT,
    tolls_amount FLOAT,
    total_amount FLOAT
) WITH (
    LOCATION = 'puYear=*/puMonth=/*/*.parquet',
    DATA_SOURCE = YellowTaxi,
    FILE_FORMAT = ParquetFormat
);
```

¹⁴ <https://github.com/Azure-Samples/Synapse/blob/master/SQL/Samples/LdwSample/SampleDB.sql>

Use an external table

You can use external tables in your queries the same way you use them in SQL server queries. The following query demonstrates this using the population external table we created in previous section. It returns country/region names with their population in 2019 in descending order.

NOTE:

Change the first line in the query, i.e. [mydbname], so you're using the database you created.

```
USE [mydbname];
GO

SELECT
    country_name, population
FROM populationExternalTable
WHERE
    [year] = 2019
ORDER BY
    [population] DESC;
```

Create views in Azure Synapse serverless SQL pools

Views will allow you to reuse queries that you create. Views are also needed if you want to use tools such as Power BI to access the data in conjunction with serverless SQL pools.

Prerequisites

Your first step is to create a database where you will execute the queries. Then initialize the objects by executing the following **setup script**¹⁵ on that database. This setup script will create the data sources, database scoped credentials, and external file formats that are used in these samples.

Create a view

You can create views in the same way you create regular SQL Server views. The following query creates a view that reads population.csv file.

NOTE:

Change the first line in the query, i.e., [mydbname], so you're using the database you created.

```
USE [mydbname];
GO

DROP VIEW IF EXISTS populationView;
GO

CREATE VIEW populationView AS
SELECT *
FROM OPENROWSET(
    BULK 'csv/population/population.csv',
```

¹⁵ <https://github.com/Azure-Samples/Synapse/blob/master/SQL/Samples/LdwSample/SampleDB.sql>

```
        DATA_SOURCE = 'SqlOnDemandDemo',
        FORMAT = 'CSV',
        FIELDTERMINATOR = ',',
        ROWTERMINATOR = '\n'
    )
WITH (
    [country_code] VARCHAR (5) COLLATE Latin1_General_BIN2,
    [country_name] VARCHAR (100) COLLATE Latin1_General_BIN2,
    [year] smallint,
    [population] bigint
) AS [r];
```

The view in this example uses the OPENROWSET function that uses an absolute path to the underlying files. If you have EXTERNAL DATA SOURCE with a root URL of your storage, you can use OPENROWSET with DATA_SOURCE and relative file path:

```
CREATE VIEW TaxiView
AS SELECT *, nyc.filepath(1) AS [year], nyc.filepath(2) AS [month]
FROM
OPENROWSET(
    BULK 'parquet/taxi/year=*/month=/*/*.parquet',
    DATA_SOURCE = 'sqlondemanddemo',
    FORMAT='PARQUET'
) AS nyc
```

Use a view

You can use views in your queries the same way you use views in SQL Server queries. The following query demonstrates using the population_csv view we created. It returns country/region names with their population in 2019 in descending order.

NOTE:

Change the first line in the query, i.e., [mydbname], so you're using the database you created.

```
USE [mydbname];
GO

SELECT
    country_name, population
FROM populationView
WHERE
    [year] = 2019
ORDER BY
    [population] DESC;
```

Knowledge check

Question 1

Which metadata object do you create that contains a query that reads data from a data lake?

- VIEW.
- EXTERNAL TABLE.
- CREDENTIAL.

Question 2

Which metadata object provides the connection information to the files in a data lake store?

- DATABASE.
- DATA SOURCE.
- EXTERNAL TABLE.

Question 3

Which CREDENTIAL identity type can be time limited?

- Azure Active Directory user.
- Managed identity.
- Shared access signature.

Summary

Using familiar T-SQL concepts like databases, external tables, and views you can easily create a virtual SQL layer over the data in the lake. This will enable any client that can connect to serverless SQL pool to interact with data residing in the data lake, without a need to move or ingest the data.

Now that you have completed this lesson, you have learned how to:

- Create databases in Azure Synapse serverless SQL pools
- Create and manage credentials in Azure Synapse serverless SQL pools
- Create external data sources in Azure Synapse serverless SQL pools
- Create external tables in Azure Synapse serverless SQL pools
- Create views in Azure Synapse serverless SQL pools

Secure data and manage users in Azure Synapse serverless SQL pools

Introduction

In this lesson, you will learn how you can set up security when using Azure Synapse serverless SQL pools

After the completion of this lesson, you will be able to:

- Choose an authentication method in Azure Synapse serverless SQL pools
- Manage users in Azure Synapse serverless SQL pools
- Manage user permissions in Azure Synapse serverless SQL pools

Prerequisites

Before taking this lesson, it is recommended that the student is able to:

- Log into the Azure portal
- Explain the different components of Azure Synapse Analytics
- Use Azure Synapse Studio

Choose an authentication method in Azure Synapse serverless SQL pools

Serverless SQL pool authentication refers to how users prove their identity when connecting to the endpoint. Two types of authentication are supported:

- **SQL Authentication**

This authentication method uses a username and password.

- **Azure Active Directory Authentication**

This authentication method uses identities managed by Azure Active Directory. For Azure AD users, multi-factor authentication can be enabled. Use Active Directory authentication (integrated security) whenever possible.

Authorization

Authorization refers to what a user can do within a serverless SQL pool database and is controlled by your user account's database role memberships and object-level permissions.

If SQL Authentication is used, the SQL user exists only in the serverless SQL pool and permissions are scoped to the objects in the serverless SQL pool. Access to securable objects in other services (such as Azure Storage) can't be granted to a SQL user directly since it only exists in scope of serverless SQL pool. The SQL user needs get authorization to access the files in the storage account.

If Azure Active Directory authentication is used, a user can sign in to a serverless SQL pool and other services, like Azure Storage, and can grant permissions to the Azure Active Directory user.

Access to storage accounts

A user that is logged into the serverless SQL pool service must be authorized to access and query the files in Azure Storage. Serverless SQL pool supports the following authorization types:

- Anonymous access

To access publicly available files placed on Azure storage accounts that allow anonymous access.

- Shared access signature (SAS)

Provides delegated access to resources in storage account. With a SAS, you can grant clients access to resources in storage account, without sharing account keys. A SAS gives you granular control over the type of access you grant to clients who have the SAS: validity interval, granted permissions, acceptable IP address range, acceptable protocol (https/http).

- Managed Identity.

Is a feature of Azure Active Directory (Azure AD) that provides Azure services for serverless SQL pool. Also, it deploys an automatically managed identity in Azure AD. This identity can be used to authorize the request for data access in Azure Storage. Before accessing the data, the Azure Storage administrator must grant permissions to Managed Identity for accessing the data. Granting permissions to Managed Identity is done the same way as granting permission to any other Azure AD user.

- User Identity

Also known as "pass-through", is an authorization type where the identity of the Azure AD user that logged into serverless SQL pool is used to authorize access to the data. Before accessing the data, Azure Storage administrator must grant permissions to Azure AD user for accessing the data. This authorization type uses the Azure AD user that logged into serverless SQL pool, therefore it's not supported for SQL user types.

Supported authorization types for database users can be found in the table below:

Authorization type	SQL user	Azure AD user
User Identity	Not supported	Supported
SAS	Supported	Supported
Managed Identity	Not supported	Supported

Supported storage and authorization types can be found in the table below:

Authorization type	Blob Storage	ADLS Gen1	ADLS Gen2
User Identity	Supported - SAS token can be used to access storage that is not protected with firewall	Not supported	Supported - SAS token can be used to access storage that is not protected with firewall
SAS	Supported	Supported	Supported
Managed Identity	Supported	Supported	Supported

Manage users in Azure Synapse serverless SQL pools

You can give administrator privileges to a user to Azure Synapse serverless SQL pool. To do this you should open the Azure Synapse workspace and do the following steps:

1. Go to **Manage** menu

2. Go to **Access control**

3. Click on **Add**

Access control

Grant others access to this workspace by assigning roles to users, groups, and/or service principals. [Learn more](#)

 Add  Refresh  Remove access

4. Choose **Synapse Administrator**

Add role assignment

Grant others access to this workspace by assigning roles to users, groups, and/or service principals.

[Learn more](#)

Scope * ⓘ

Workspace Workspace item

Role * ⓘ

Synapse Administrator

Filter...

Synapse Administrator ⓘ

Synapse SQL Administrator ⓘ

Synapse Apache Spark Administrator ⓘ

Synapse Contributor (preview) ⓘ

Synapse Artifact Publisher (preview) ⓘ

Synapse Artifact User (preview) ⓘ

Synapse Compute Operator (preview) ⓘ

Synapse Credential User (preview) ⓘ

5. Select a User or Security group (a security group is the recommended option here)

6. Click **Apply**

Now this user or group is the administrator of the Azure Synapse workspace and serverless SQL pool.

Manage user permissions in Azure Synapse serverless SQL pools

To secure data, Azure Storage implements an access control model that supports both Azure role-based access control (Azure RBAC) and Portable Operating System Interface for Unix (POSIX) like access control lists (ACLs).

You can associate a security principal with an access level for files and directories. These associations are captured in an access control list (ACL). Each file and directory in your storage account has an access control list. When a security principal attempts an operation on a file or directory, An ACL check determines whether that security principal (user, group, service principal, or managed identity) has the correct permission level to perform the operation.

There are two kinds of access control lists:

- **Access ACLs**

Controls access to an object. Files and directories both have access ACLs.

- **Default ACLs**

Are templates of ACLs associated with a directory that determine the access ACLs for any child items that are created under that directory. Files do not have default ACLs.

Both access ACLs and default ACLs have the same structure.

The permissions on a container object are Read, Write, and Execute, and they can be used on files and directories as shown in the following table:

Levels of permissions

Permission	File	Directory
Read (R)	Can read the contents of a file	Requires Read and Execute to list the contents of the directory
Write (W)	Can write or append to a file	Requires Write and Execute to create child items in a directory
Execute (X)	Does not mean anything in the context of Data Lake Storage Gen2	Required to traverse the child items of a directory

Guidelines in setting up ACLs

Always use Azure Active Directory security groups as the assigned principal in an ACL entry. Resist the opportunity to directly assign individual users or service principals. Using this structure will allow you to add and remove users or service principals without the need to reapply ACLs to an entire directory structure. Instead, you can just add or remove users and service principals from the appropriate Azure AD security group.

There are many ways to set up groups. For example, imagine that you have a directory named **/LogData** which holds log data that is generated by your server. Azure Data Factory (ADF) ingests data into that folder. Specific users from the service engineering team will upload logs and manage other users of this folder, and various Databricks clusters will analyze logs from that folder.

To enable these activities, you could create a LogsWriter group and a LogsReader group. Then, you could assign permissions as follows:

- Add the LogsWriter group to the ACL of the **/LogData** directory with rwx permissions.

- Add the LogsReader group to the ACL of the **/LogData** directory with r-x permissions.
- Add the service principal object or Managed Service Identity (MSI) for ADF to the LogsWriters group.
- Add users in the service engineering team to the LogsWriter group.
- Add the service principal object or MSI for Databricks to the LogsReader group.

If a user in the service engineering team leaves the company, you could just remove them from the LogsWriter group. If you did not add that user to a group, but instead, you added a dedicated ACL entry for that user, you would have to remove that ACL entry from the **/LogData** directory. You would also have to remove the entry from all subdirectories and files in the entire directory hierarchy of the **/LogData** directory.

Roles necessary for serverless SQL pool users

For users which need **read only** access you should assign role named **Storage Blob Data Reader**.

For users which need **read/write** access you should assign role named **Storage Blob Data Contributor**. Read/Write access is needed if user should have access to create external table as select (CETAS).

NOTE:

If user has a role Owner or Contributor, that role is not enough. Azure Data Lake Storage gen 2 has super-roles which should be assigned.

Database level permission

To provide more granular access to the user, you should use Transact-SQL syntax to create logins and users.

To grant access to a user to a single serverless SQL pool database, follow the steps in this example:

1. Create LOGIN

```
use master  
CREATE LOGIN [alias@domain.com] FROM EXTERNAL PROVIDER;
```

2. Create USER

```
use yourdb -- Use your DB name  
CREATE USER alias FROM LOGIN [alias@domain.com];
```

3. Add USER to members of the specified role

```
use yourdb -- Use your DB name  
alter role db_datareader  
Add member alias -- Type USER name from step 2  
-- You can use any Database Role which exists  
-- (examples: db_owner, db_datareader, db_datawriter)  
-- Replace alias with alias of the user you would like to give access and domain with the company  
domain you are using.
```

Server level permission

1. To grant full access to a user to all serverless SQL pool databases, follow the step in this example:

```
CREATE LOGIN [alias@domain.com] FROM EXTERNAL PROVIDER;  
ALTER SERVER ROLE sysadmin ADD MEMBER [alias@domain.com];
```

Knowledge check

Question 1

Which authentication method would be the likeliest choice to use for an individual who needs to access your serverless SQL pool who works for an external organization?

- Local authentication.
- SQL Authentication.
- Azure Active Directory.

Question 2

Which Azure Synapse Studio hub is where you assign administrator privileges to an Azure Synapse workspace?

- Manage.
- Data.
- Develop.

Question 3

Which role enables a user to create external table as select (CETAS) against an Azure Data Lake Gen2 data store?

- Storage Blob Data Reader.
- Storage Blob Data Contributor.
- Executor.

Summary

In this lesson, you will learn how you can set up security when using Azure Synapse serverless SQL pools by:

- Choosing an authentication method in Azure Synapse serverless SQL pools
- Managing users in Azure Synapse serverless SQL pools
- Managing user permissions in Azure Synapse serverless SQL pools

Module Lab information

Lab 2 - Run interactive queries using Azure Synapse Analytics serverless SQL pools

- **Estimated Time:** 50 - 60 minutes
- **Lab files:** The files for this lab are located in the *Instructions\Labs\04* folder.

Lab overview

In this lab, students will learn how to work with files stored in the data lake and external file sources, through T-SQL statements executed by a serverless SQL pool in Azure Synapse Analytics. Students will query Parquet files stored in a data lake, as well as CSV files stored in an external data store. Next, they will create Azure Active Directory security groups and enforce access to files in the data lake through Role-Based Access Control (RBAC) and Access Control Lists (ACLs).

Lab objectives

After completing this lab, you will be able to:

- Query Parquet data with serverless SQL pools
- Create external tables for Parquet and CSV files
- Create views with serverless SQL pools
- Secure access to data in a data lake when using serverless SQL pools
- Configure data lake security using Role-Based Access Control (RBAC) and Access Control Lists (ACLs)

Module summary

module-summary

In this module, you have learned how to query and prepare data in an interactive way on files placed in Azure Storage using Azure Synapse Analytics.

Learning objectives

In this module, you have:

- Explored Azure Synapse serverless SQL pools capabilities
- Queried data in the lake using Azure Synapse serverless SQL pools
- Created metadata objects in Azure Synapse serverless SQL pools
- Secured data and managed users in Azure Synapse serverless SQL pools

Post Course Review

After the course, consider reading the documentation on the **best practices for serverless SQL pool in Azure Synapse Analytics¹⁶**.

Important

Remember

Should you be working in your own subscription while running through this content, it is best practice at the end of a project to identify whether or not you still need the resources you created. Resources left running can cost you money.

You can delete resources one by one, or just delete the resource group to get rid of the entire set.

¹⁶ <https://docs.microsoft.com/en-us/azure/synapse-analytics/sql/best-practices-serverless-sql-pool>

Answers

Question 1

Performing ad hoc data exploration in Azure Synapse Analytics uses which resource model?

- Dedicated SQL pools.
- Serverless SQL pools.
- Serverless Spark pools.

Explanation

Correct. Serverless SQL pools are used to perform ad hoc data exploration in Azure Synapse Analytics.

Question 2

What objects are used to provide a SQL metadata layer that creates a logical data warehouse in Azure Synapse Analytics?

- Views.
- Stored procedures.
- Functions.

Explanation

Correct. Views are used to provide a SQL metadata layer that creates a logical data warehouse.

Question 3

Which Azure storage service is the primary service used to query data from using Serverless SQL pools?

- Azure SQL Database.
- Azure Cosmos DB.
- Azure Data Lake.

Explanation

Correct. Azure Data Lake is the primary service used to query data from using Serverless SQL pools.

Question 1

What function is used to read the data in files stored in a data lake?

- FORMAT.
- ROWSET.
- OPENROWSET.

Explanation

Correct. The OPENROWSET is used to read the data in files stored in a data lake.

Question 2

What value should be set in the fieldterminator and fieldquote variables to read JSON files?

- 0b0x.
- 00xb.
- 0x0b.

Explanation

Correct. 0x0b is the value should be set in the fieldterminator and fieldquote variables to read JSON files.

Question 3

What character in file path can be used to select all the file/folders that match rest of the path?

- &.
- *.
- /.

Explanation

Correct. The asterisk character in file path can be used to select all the file or folders that match rest of the path.

Question 1

Which metadata object do you create that contains a query that reads data from a data lake?

- VIEW.
- EXTERNAL TABLE.
- CREDENTIAL.

Explanation

Correct. A VIEW is a metadata object that contains a query that reads data from a data lake.

Question 2

Which metadata object provides the connection information to the files in a data lake store?

- DATABASE.
- DATA SOURCE.
- EXTERNAL TABLE.

Explanation

Correct. A DATA SOURCE provides the connection information to the files in a data lake store.

Question 3

Which CREDENTIAL identity type can be time limited?

- Azure Active Directory user.
- Managed identity.
- Shared access signature.

Explanation

Correct. A Shared access signature can be set to a time limit.

Question 1

Which authentication method would be the likeliest choice to use for an individual who needs to access your serverless SQL pool who works for an external organization?

- Local authentication.
- SQL Authentication.
- Azure Active Directory.

Explanation

Correct. SQL Authentication uses an authentication method of a username and password stored within the serverless SQL pool.

Question 2

Which Azure Synapse Studio hub is where you assign administrator privileges to an Azure Synapse workspace?

- Manage.
- Data.
- Develop.

Explanation

Correct. The manage hub is the area of Azure Synapse Studio where you assign administrator privileges to an Azure Synapse workspace.

Question 3

Which role enables a user to create external table as select (CETAS) against an Azure Data Lake Gen2 data store?

- Storage Blob Data Reader.
- Storage Blob Data Contributor.
- Executor.

Explanation

Correct. This provides the read/write access required to execute a create external table as select statement.

Module 3 Data Exploration and Transformation in Azure Databricks

Module introduction

module-introduction

In this module, students will learn how to harness the power of Apache Spark and powerful clusters running on the Azure Databricks platform to run large data engineering workloads in the cloud combined with Azure Synapse Analytics.

Learning objectives

In this module, you will:

- Describe Azure Databricks
- Read and write data in Azure Databricks
- Work with DataFrames in Azure Databricks
- Work with DataFrames advanced methods in Azure Databricks

Describe Azure Databricks

Introduction

Azure Databricks is a fully-managed version of the open-source **Apache Spark¹** analytics and data processing engine. Azure Databricks is an enterprise-grade and secure cloud-based big data and machine learning platform.

Databricks provides a notebook-oriented Apache Spark as-a-service workspace environment, making it easy to manage clusters and explore data interactively.

Learning objectives

In this module, you will:

- Understand the Azure Databricks platform
- Create your own Azure Databricks workspace
- Create a notebook inside your home folder in Databricks
- Understand the fundamentals of Apache Spark notebook
- Create, or attach to, a Spark cluster
- Identify the types of tasks well-suited to the unified analytics engine Apache Spark

Prerequisites

None

Explain Azure Databricks

Azure Databricks is a fully-managed, cloud-based Big Data and Machine Learning platform, which empowers developers to accelerate AI and innovation by simplifying the process of building enterprise-grade production data applications. Built as a joint effort by the team that started Apache Spark and Microsoft, Azure Databricks provides data science and engineering teams with a single platform for Big Data processing and Machine Learning.

By combining the power of Databricks, an end-to-end, managed Apache Spark platform optimized for the cloud, with the enterprise scale and security of Microsoft's Azure platform, Azure Databricks makes it simple to run large-scale Spark workloads.

Optimized environment

To address the problems seen on other Big Data platforms, Azure Databricks was optimized from the ground up, with a focus on performance and cost-efficiency in the cloud. The Databricks Runtime adds several key capabilities to Apache Spark workloads that can increase performance and reduce costs by as much as 10-100x when running on Azure, including:

- High-speed connectors to Azure storage services, such as Azure Blob Store and Azure Data Lake
- Auto-scaling and auto-termination of Spark clusters to minimize costs

¹ <https://spark.apache.org/>

- Caching
- Indexing
- Advanced query optimization

By providing an optimized, easy to provision and configure environment, Azure Databricks gives developers a performant, cost-effective platform that enables them to spend more time building applications, and less time focused on managing clusters and infrastructure.

Who is Databricks?

Databricks was founded by the creators of Apache Spark, Delta Lake, and MLflow.

Over 2000 global companies use the Databricks platform across big data & machine learning lifecycle.

Databricks Vision: Accelerate innovation by unifying data science, data engineering and business.

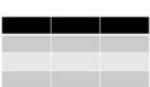
Databricks Solution: Big Data Analytics Platform

What does Databricks offer that is not Open-Source Spark?

- Databricks Workspace - Interactive Data Science & Collaboration
- Databricks Workflows - Production Jobs & Workflow Automation
- Databricks Runtime
- Databricks I/O (DBIO) - Optimized Data Access Layer
- Databricks Serverless - Fully Managed Auto-Tuning Platform
- Databricks Enterprise Security (DBES) - End-To-End Security & Compliance

What is Apache Spark?

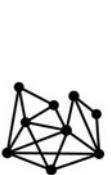
Spark is a unified processing engine that can analyze big data using SQL, machine learning, graph processing, or real-time stream analysis:



SQL / DataFrames



Machine Learning



Graph



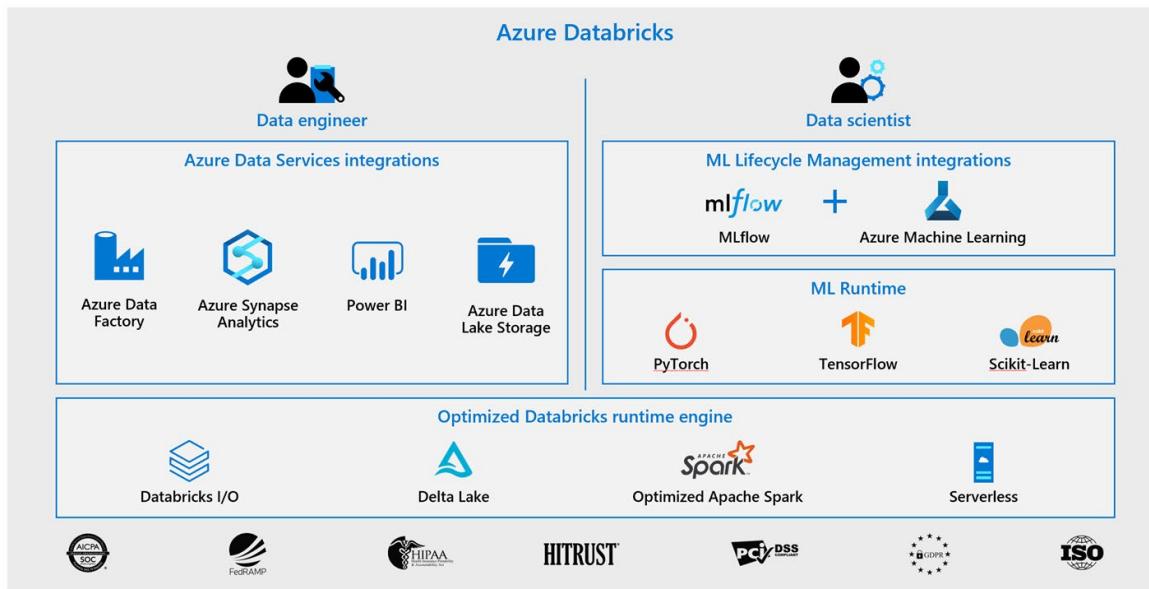
Streaming

- At its core is the Spark Engine.
- The DataFrames API provides an abstraction above RDDs while simultaneously improving performance 5-20x over traditional RDDs with its Catalyst Optimizer.

- Spark ML provides high quality and finely tuned machine learning algorithms for processing big data.
- The Graph processing API gives us an easily approachable API for modeling pairwise relationships between people, objects, or nodes in a network.
- The Streaming APIs give us End-to-End Fault Tolerance, with Exactly-Once semantics, and the possibility for sub-millisecond latency.

And it all works together seamlessly!

Azure Databricks



As a compute engine, Azure Databricks sits at the center of your Azure-based software platform and provides native integration with Azure Active Directory (Azure AD) and other Azure services.

Scala, Python, Java, R & SQL

- Besides being able to run in many environments, Apache Spark makes the platform even more approachable by supporting multiple languages:
 - Scala - Apache Spark's primary language
 - Python - More commonly referred to as PySpark
 - R - **SparkR²** (R on Spark)
 - Java
 - SQL - Closer to ANSI SQL 2003 compliance
 - Now running all 99 TPC-DS queries
 - New standards-compliant parser (with good error messages!)

² <https://spark.apache.org/docs/latest/sparkr.html>

- Subqueries (correlated & uncorrelated)
- Approximate aggregate stats
- With the DataFrames API, the performance differences between languages are nearly nonexistence (especially for Scala, Java & Python).

Create an Azure Databricks workspace and cluster

When talking about the Azure Databricks workspace, we refer to two different things. The first reference is the logical Azure Databricks environment in which clusters are created, data is stored (via DBFS), and in which the server resources are housed. The second reference is the more common one used within the context of Azure Databricks. That is the special root folder for all of your organization's Databricks assets, including notebooks, libraries, and dashboards, as shown below:

The screenshot shows the Microsoft Azure portal interface. On the left, there is a vertical sidebar with icons for Azure Databricks, Home, Workspace, Recents, Data, Clusters, Jobs, and Search. The 'Workspace' icon is highlighted. The main area has a dark header with the text 'Microsoft Azure'. Below the header, the word 'Workspace' is displayed. A dropdown menu is open under 'Workspace', showing 'Shared' and 'Users'. The 'Users' option is selected. To the right of this, another dropdown menu is open under 'Users', listing 16 entries, each starting with 'databricks' followed by a number and '@'. The first entry, 'joel@...', is highlighted. A large circular button with a right-pointing arrow is positioned to the right of the user list.

The first step to using Azure Databricks is to create and deploy a Databricks workspace, which is the logical environment. You can do this in the Azure portal.

Deploy an Azure Databricks workspace

1. Open the Azure portal.
2. Click **Create a Resource** in the top left
3. Search for "Databricks"
4. Select *Azure Databricks*

5. On the Azure Databricks page select **Create**
6. Provide the required values to create your Azure Databricks workspace:
 - **Subscription:** Choose the Azure subscription in which to deploy the workspace.
 - **Resource Group:** Use **Create new** and provide a name for the new resource group.
 - **Location:** Select a location near you for deployment. For the list of regions that are supported by Azure Databricks, see **Azure services available by region**³.
 - **Workspace Name:** Provide a unique name for your workspace.
 - **Pricing Tier: Trial (Premium - 14 days Free DBUs)**. You must select this option when creating your workspace or you will be charged. The workspace will suspend automatically after 14 days. When the trial is over you can convert the workspace to **Premium** but then you will be charged for your usage.
7. Select **Review + Create**.
8. Select **Create**.

The workspace creation takes a few minutes. During workspace creation, the **Submitting deployment for Azure Databricks** tile appears on the right side of the portal. You might need to scroll right on your dashboard to see the tile. There's also a progress bar displayed near the top of the screen. You can watch either area for progress.

What is a cluster?

The notebooks are backed by clusters, or networked computers, that work together to process your data. The first step is to create a cluster.

Create a cluster

1. When your Azure Databricks workspace creation is complete, select the link to go to the resource.
2. Select **Launch Workspace** to open your Databricks workspace in a new tab.
3. In the left-hand menu of your Databricks workspace, select **Clusters**.
4. Select **Create Cluster** to add a new cluster.

³ <https://azure.microsoft.com/regions/services/>

Create Cluster

New Cluster | [Cancel](#) | [Create Cluster](#) | 2-8 Workers: 28.0-112.0 GB Memory, 8-32 Cores, 0.75 DBU
1 Driver: 14.0 GB Memory, 4 Cores, 0.75 DBU

Cluster Name: Test Cluster

Cluster Mode: Standard

Pool: None

Databricks Runtime Version: [Learn more](#)
Runtime: 6.4 (Scala 2.11, Spark 2.4.5)

New This Runtime version supports only Python 3.

Autopilot Options:
 Enable autoscaling
 Terminate after 120 minutes of inactivity

Worker Type	Min Workers	Max Workers	
Standard_DS3_v2	14.0 GB Memory, 4 Cores, 0.75 DBU	2	8

Driver Type: Same as worker

5. Enter a name for your cluster. Use your name or initials to easily differentiate your cluster from your coworkers.
 6. Select the **Databricks RuntimeVersion**. We recommend the latest runtime and **Scala 2.11**.
 7. Specify your cluster configuration. While on the 14 day free trial, the defaults will be sufficient. When the trial is ended, you may prefer to change **Min Workers** to zero. That will allow the compute resources to shut down when you are not in a coding exercise and reduce your charges.
- Hint:** Check with your local system administrator to see if there is a recommended default cluster at your company to use for the rest of the class. This could save you some money!
8. Select **Create Cluster**.

Understand Azure Databricks Notebooks

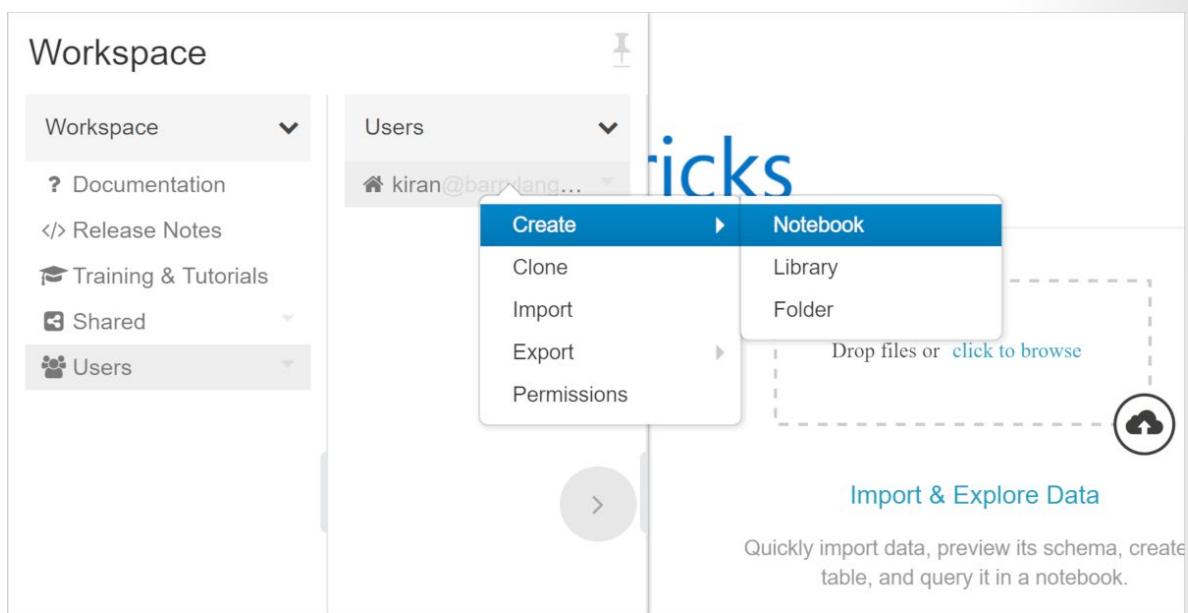
After creating your Databricks workspace, it's time to create your first notebook. To execute your notebook, you will attach the cluster you created in the previous unit.

What is Apache Spark notebook?

A notebook is a collection of cells. These cells are run to execute code, to render formatted text, or to display graphical visualizations.

Create a notebook

1. In the Azure portal, click **All resources** menu on the left side navigation and select the Databricks workspace you created in the last unit.
2. Select **Launch Workspace** to open your Databricks workspace in a new tab.
3. On the left-hand menu of your Databricks workspace, select **Home**.
4. Right-click on your home folder.
5. Select **Create**.
6. Select **Notebook**.



7. Name your notebook **First Notebook**.
8. Set the **Language** to **Python**.
9. Select the cluster to which to attach this notebook.

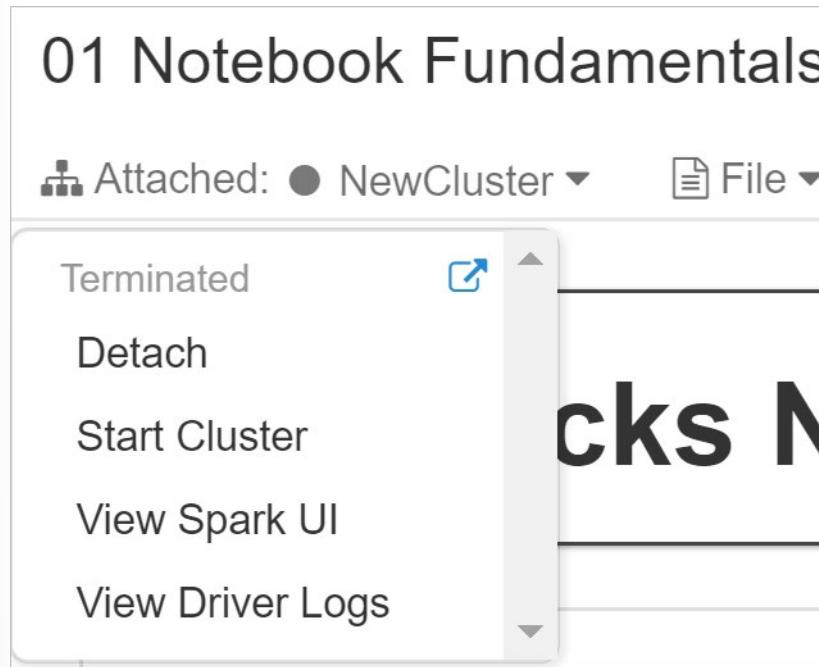
NOTE: This option displays only when a cluster is currently running. You can still create your notebook and attach it to a cluster later.

10. Select **Create**.

Now that you've created your notebook, let's use it to run some code.

Attach and detach your notebook

To use your notebook to run a code, you must attach it to a cluster. You can also detach your notebook from a cluster and attach it to another depending upon your organization's requirements.



If your notebook is attached to a cluster, you can:

- Detach your notebook from the cluster
- Restart the cluster
- Attach to another cluster
- Open the Spark UI
- View the log files of the driver

Exercise: Work with Notebooks

You can use Apache Spark notebooks to:

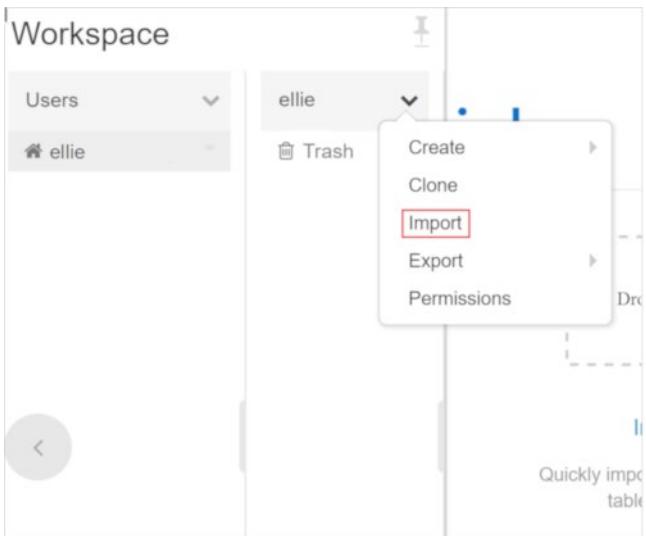
- Read and process huge files and data sets
- Query, explore, and visualize data sets
- Join disparate data sets found in data lakes
- Train and evaluate machine learning models
- Process live streams of data
- Perform analysis on large graph data sets and social networks

To learn more about using notebooks, clone the labs archive where sample notebooks are provided. These notebooks will help you understand how to use notebooks for your day-to-day tasks.

Clone the Databricks archive

1. In the Azure portal, navigate to your deployed Azure Databricks workspace and select **Launch Workspace**.
2. In the left pane, select **Workspace > Users**, and select your username (the entry with the house icon).

3. In the pane that appears, select the arrow next to your name, and select **Import**.



4. In the **Import Notebooks** dialog box, select the URL and paste in the following URL:

<https://github.com/solliancenet/microsoft-learning-paths-databricks-notebooks/blob/master/data-engineering/DBC/01-Introduction-to-Azure-Databricksdbc?raw=true>

1. Select **Import**.
2. Select the **01-Introduction-to-Azure-Databricks** folder that appears.
3. Use the set of notebooks in this folder to complete this lab.

Complete the following notebook

- **01-The-Databricks-Environment** - This notebook illustrates the fundamentals of a Databricks notebook.

Knowledge check

Question 1

How many drivers does a Cluster have?

- Only one.
- Two, running in parallel.
- Configurable between one and eight.

Question 2

Spark is a distributed computing environment. Therefore, work is parallelized across executors. At which two levels does this parallelization occur?

- The Executor and the Slot.
- The Driver and the Executor.
- The Slot and the Task.

Question 3

What type of process are the driver and the executors?

- Java processes.
- Python processes.
- C++ processes.

Summary

In this module, you learned the basics about using Databricks workspace and Apache Spark notebooks. The notebooks allow you to interactively work with different types of data. You can use notebooks to process huge data files, query, read and write data from different sources, train machine learning models, and process live data streams.

Now that you have concluded this module, you should know:

1. How to Create a Cluster
2. How to Attach a Notebook to a Cluster
3. How to Execute Python, Scala, SQL or R code in a Notebook Cell
4. How to Detach a Notebook from a Cluster
5. How Azure Databricks fits into the Azure Ecosystem
6. How DBFS works to present Blob Storage as a Filesystem

Clean up

If you plan on completing other Azure Databricks modules, don't delete your Azure Databricks instance yet. You can use the same environment for the other modules.

Delete the Azure Databricks instance

1. Navigate to the Azure portal.
2. Navigate to the resource group that contains your Azure Databricks instance.
3. Select **Delete resource group**.
4. Type the name of the resource group in the confirmation text box.
5. Select **Delete**.

Read and write data in Azure Databricks

Introduction

Suppose you're working for a data analytics startup that's now expanding along with its increasing customer base. You receive customer data from multiple sources in different raw formats. To efficiently handle huge amounts of customer data, your company has decided to invest in Azure Databricks. Your team is responsible for analyzing how Databricks supports day-to-day data-handling functions, such as reads, writes, and queries. Your team performs these tasks to prepare the data for advanced analytics and machine learning operations.

Learning objectives

In this module, you'll:

- Use Azure Databricks to read multiple file types, both with and without a Schema.
- Combine inputs from files and data stores, such as Azure SQL Database.
- Transform and store that data for advanced analytics.

Prerequisites

- An Azure subscription. If you don't have an Azure subscription, create a [free account](#)⁴.

Read data in CSV format

In this unit, you need to complete the exercises within a Databricks Notebook. To begin, you need to have access to an Azure Databricks workspace. If you do not have a workspace available, follow the instructions below. Otherwise, you can skip to the bottom of the page to Clone the Databricks archive.

Unit Pre-requisites

Microsoft Azure Account: You will need a valid and active Azure account for the Azure labs. If you do not have one, you can sign up for a [free trial](#)⁵

- If you are a Visual Studio Active Subscriber, you are entitled to Azure credits per month. You can refer to this [link](#)⁶ to find out more including how to activate and start using your monthly Azure credit.
- If you are not a Visual Studio Subscriber, you can sign up for the FREE **Visual Studio Dev Essentials**⁷ program to create Azure free account.

Create the required resources

To complete this lab, you will need to deploy an Azure Databricks workspace in your Azure subscription.

⁴ <https://azure.microsoft.com/free>

⁵ <https://azure.microsoft.com/free/>

⁶ <https://azure.microsoft.com/pricing/member-offers/msdn-benefits-details/>

⁷ <https://www.visualstudio.com/dev-essentials/>

Deploy an Azure Databricks workspace

1. Click the following button to open the Azure Resource Manager Template (ARM) template in the Azure portal.
Deploy Databricks from the ARM Template⁸
2. Provide the required values to create your Azure Databricks workspace:
 - **Subscription:** Choose the Azure Subscription in which to deploy the workspace.
 - **Resource Group:** Leave at Create new and provide a name for the new resource group.
 - **Location:** Select a location near you for deployment. For the list of regions supported by Azure Databricks, see **Azure services available by region⁹**.
 - **Workspace Name:** Provide a name for your workspace.
 - **Pricing Tier:** Ensure premium is selected.
3. Accept the terms and conditions.
4. Select Purchase.
5. The workspace creation takes a few minutes. During workspace creation, the portal displays the Submitting deployment for Azure Databricks tile on the right side. You may need to scroll right on your dashboard to see the tile. There is also a progress bar displayed near the top of the screen. You can watch either area for progress.

Create a cluster

1. When your Azure Databricks workspace creation is complete, select the link to go to the resource.
2. Select **Launch Workspace** to open your Databricks workspace in a new tab.
3. In the left-hand menu of your Databricks workspace, select **Clusters**.
4. Select **Create Cluster** to add a new cluster.

⁸ <https://portal.azure.com/#create/Microsoft.Template/uri/https%3A%2F%2Fraw.githubusercontent.com%2FAzure%2Fazure-quickstart-templates%2Fmaster%2F101-databricks-workspace%2Fazuredeploy.json>

⁹ <https://azure.microsoft.com/regions/services/>

Create Cluster

New Cluster | Cancel | Create Cluster | 2-8 Workers: 28.0-112.0 GB Memory, 8-32 Cores, 1.5-6 DBU
1 Driver: 14.0 GB Memory, 4 Cores, 0.75 DBU ?

Cluster Name: Test Cluster

Cluster Mode: Standard

Pool: None

Databricks Runtime Version: Runtime: 8.0 (Scala 2.12, Spark 3.1.1) | Learn more

Note: Databricks Runtime 8.x uses Delta Lake as the default table format. [Learn more](#)

Autopilot Options:

Enable autoscaling ?
 Terminate after 120 minutes of inactivity ?

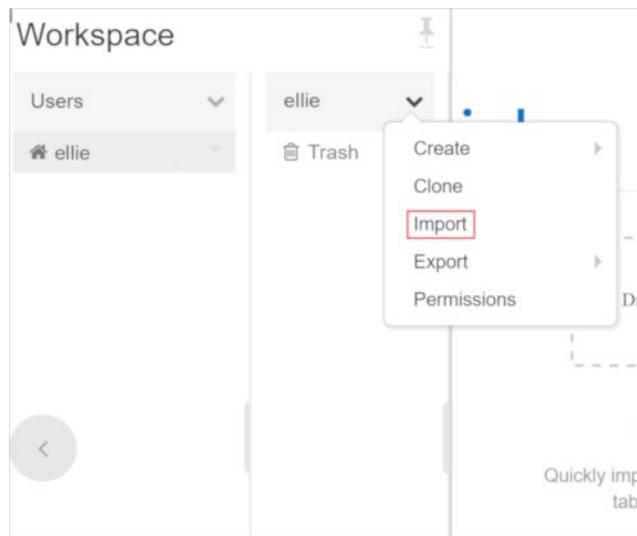
Worker Type: Standard_DS3_v2 | 14.0 GB Memory, 4 Cores, 0.75 DBU | Min Workers: 2 | Max Workers: 8 | Spot instances ?

Driver Type: Same as worker | 14.0 GB Memory, 4 Cores, 0.75 DBU |

5. Enter a name for your cluster. Use your name or initials to easily differentiate your cluster from your coworkers.
6. Select the **Databricks RuntimeVersion**. We recommend the latest runtime and **Scala 2.12**.
7. Select the default values for the cluster configuration.
8. Select **Create Cluster**.

Clone the Databricks archive

1. If you do not currently have your Azure Databricks workspace open: in the Azure portal, navigate to your deployed Azure Databricks workspace and select **Launch Workspace**.
2. In the left pane, select **Workspace > Users**, and select your username (the entry with the house icon).
3. In the pane that appears, select the arrow next to your name, and select **Import**.



4. In the **Import Notebooks** dialog box, select the URL and paste in the following URL:

<https://github.com/solliancenet/microsoft-learning-paths-databricks-notebooks/blob/master/data-engineering/DBC/03-Reading-and-writing-data-in-Azure-Databricksdbc?raw=true>

1. Select **Import**.
2. Select the **03-Reading-and-writing-data-in-Azure-Databricks** folder that appears.

Complete the following notebook

Open the **1.Reading Data - CSV** notebook. Make sure you attach your cluster to the notebook before following the instructions and running the cells within.

Within the notebook, you will:

- Start working with the API documentation
- Introduce the class `SparkSession` and other entry points
- Introduce the class `DataFrameReader`
- Read data from:
 - CSV without a Schema
 - CSV with a Schema

Read the CSV file

In the notebook, you start by reading the CSV file by specifying the tab character as the delimiter and the location of the file:

```
# A reference to our tab-separated-file  
csvFile = "/mnt/training/wikipedia/pageviews/pageviews_by_second.tsv"  
  
tempDF = (spark.read  
          # The DataFrameReader
```

```

    .option("sep", "\t")           # Use tab delimiter (default is comma-sep-
    arator)
    .csv(csvFile)                # Creates a DataFrame from CSV after reading
in the file
)

```

This is guaranteed to trigger one job.

A **Job** is triggered anytime we are “physically” **required to touch the data**.

In some cases, **one action may create multiple jobs** (multiple reasons to touch the data).

In this case, the reader has to “**peek**” at the first line of the file to determine how many columns of data we have.

We can see the structure of the DataFrame by executing the command `printSchema()`

It prints to the console the name of each column, its data type and if it's null or not.

```
tempDF.printSchema()
```

This command displays the following schema:

```

root
|-- _c0: string (nullable = true)
|-- _c1: string (nullable = true)
|-- _c2: string (nullable = true)

```

We can see from the schema that...

- there are three columns
- the column names **_c0**, **_c1**, and **_c2** (automatically generated names)
- all three columns are **strings**
- all three columns are **nullable**

Use the file's header and infer the schema

Since the file contains a header, we can add an option that tells the reader that the data contains a header and to use that header to determine the column names, using `.option("header", "true")`. We can also add an option that tells the reader to infer each column data type (the schema), by using `.option("inferSchema", "true")`.

```

(spark.read
  .option("header", "true")          # The DataFrameReader
  .option("sep", "\t")               # Use first line of all files as header
  .option("sep", "\t")               # Use tab delimiter (default is com-
ma-separator)
  .option("inferSchema", "true")     # Automatically infer data types
  .csv(csvFile)                   # Creates a DataFrame from CSV after
reading in the file
  .printSchema()
)

```

Read the CSV with a user-defined schema

This time we are going to read the same file. The difference here is that we define the schema beforehand and hopefully avoid the execution of any extra jobs.

To start, declare the schema. This is just a list of field names and data types:

```
# Required for StructField, StringType, IntegerType, etc.  
from pyspark.sql.types import *  
  
csvSchema = StructType([  
    StructField("timestamp", StringType(), False),  
    StructField("site", StringType(), False),  
    StructField("requests", IntegerType(), False)  
])
```

Now that we have a defined schema, we can run the following to read in the data and print the schema. The `schema(...)` command is where we pass the `StructType(schema)`:

```
(spark.read  
    .option('header', 'true')      # The DataFrameReader  
    .option('sep', "\t")          # Ignore line #1 - it's a header  
    .schema(csvSchema)           # Use tab delimiter (default is comma-separated)  
    .csv(csvFile)                # Use the specified schema  
    .in the file                 # Creates a DataFrame from CSV after reading  
    .printSchema()  
)
```

Continue to the next step

After you've completed the notebook, return to this screen, and continue to the next step.

Read data in JSON format

In your Azure Databricks workspace, open the **03-Reading-and-writing-data-in-Azure-Databricks** folder that you imported within your user folder.

Open the **2.Reading Data - JSON** notebook. Make sure you attach your cluster to the notebook before following the instructions and running the cells within.

Within the notebook, you will:

- Read data from:
 - JSON without a Schema
 - JSON with a Schema

Read from JSON with InferSchema

In this notebook, you start by looking at all of the different options that go along with reading in JSON files.

JSON Lines

Much like the CSV reader, the JSON reader also assumes...

- That there is one JSON object per line and...
- That it's delineated by a new-line.

This format is referred to as **JSON Lines** or **newline-delimited JSON**

More information about this format can be found at <http://jsonlines.org>¹⁰.

Read the JSON file

The command to read in JSON looks very similar to that of CSV. In addition to reading the JSON file, we will also print the resulting schema.

```
jsonFile = "dbfs:/mnt/training/wikipedia/edits/snapshot-2016-05-26.json"

wikiEditsDF = (spark.read              # The DataFrameReader
               .option("inferSchema", "true")   # Automatically infer data types &
               column names
               .json(jsonFile)                # Creates a DataFrame from JSON after
               reading in the file
               )
wikiEditsDF.printSchema()
```

With our DataFrame created, we can now take a peak at the data. But to demonstrate a unique aspect of JSON data (or any data with embedded fields), we will first create a temporary view and then view the data via SQL:

```
# create a view called wiki_edits
wikiEditsDF.createOrReplaceTempView("wiki_edits")
```

With the temporary view in place, look at the data with a simple SQL SELECT statement:

```
%sql

SELECT * FROM wiki_edits
```

Since the geocoding field has embedded data, we can reference the sub-fields directly in the following way:

```
%sql

SELECT channel, page, geocoding.city, geocoding.latitude, geocoding.longitude
FROM wiki_edits
WHERE geocoding.city IS NOT NULL
```

¹⁰ <http://jsonlines.org>

Read from JSON with a user-defined schema

To avoid the extra job, we can (just like we did with CSV) specify the schema for the DataFrame. Compared to our CSV example, the structure of this data is a little more complex.

Note that we can support complex data types as seen in the field geocoding:

```
# Required for StructField, StringType, IntegerType, etc.
from pyspark.sql.types import *

jsonSchema = StructType([
    StructField("channel", StringType(), True),
    StructField("comment", StringType(), True),
    StructField("delta", IntegerType(), True),
    StructField("flag", StringType(), True),
    StructField("geocoding", StructType([
        StructField("city", StringType(), True),
        StructField("country", StringType(), True),
        StructField("countryCode2", StringType(), True),
        StructField("countryCode3", StringType(), True),
        StructField("stateProvince", StringType(), True),
        StructField("latitude", DoubleType(), True),
        StructField("longitude", DoubleType(), True)
    ]), True),
    StructField("isAnonymous", BooleanType(), True),
    StructField("isNewPage", BooleanType(), True),
    StructField("isRobot", BooleanType(), True),
    StructField("isUnpatrolled", BooleanType(), True),
    StructField("namespace", StringType(), True),
    StructField("page", StringType(), True),
    StructField("pageURL", StringType(), True),
    StructField("timestamp", StringType(), True),
    StructField("url", StringType(), True),
    StructField("user", StringType(), True),
    StructField("userURL", StringType(), True),
    StructField("wikipediaURL", StringType(), True),
    StructField("wikipedia", StringType(), True)
])
```

This is a fairly complex schema, which takes a bit more effort to create. For a small file, manually creating the the schema may not be worth the effort. However, for a large file, the time to manually create the schema may be worth the trade off of a really long infer-schema process.

Next, we will read in the JSON file and once again print its schema:

```
(spark.read          # The DataFrameReader
 .schema(jsonSchema) # Use the specified schema
 .json(jsonFile)     # Creates a DataFrame from JSON after reading in the
 file
 .printSchema()
)
```

Continue to the next step

After you've completed the notebook, return to this screen, and continue to the next step.

Read data in Parquet format

In your Azure Databricks workspace, open the **03-Reading-and-writing-data-in-Azure-Databricks** folder that you imported within your user folder.

Open the **3.Reading Data - Parquet** notebook. Make sure you attach your cluster to the notebook before following the instructions and running the cells within.

Within the notebook, you will:

- Introduce the Parquet file format
- Read data from:
 - Parquet files without a schema
 - Parquet files with a schema

Read from Parquet files

According to the **Apache Parquet project site¹¹**, "Apache Parquet is a columnar storage format available to any project in the Hadoop ecosystem, regardless of the choice of data processing framework, data model or programming language."

About Parquet files

- Free & Open Source.
- Increased query performance over row-based data stores.
- Provides efficient data compression.
- Designed for performance on large data sets.
- Supports limited schema evolution.
- Is a splittable "file format".
- A **Column-Oriented¹²** data store

Row Format

Column Format

See also

- [https://parquet.apache.org¹³](https://parquet.apache.org)

¹¹ <https://parquet.apache.org/>

¹² https://en.wikipedia.org/wiki/Column-oriented_DBMS

¹³ <https://parquet.apache.org/>

- https://en.wikipedia.org/wiki/Apache_Parquet

Read in the Parquet files

In this notebook, you read in the Parquet files by specifying the location of the parquet directory and using the `parquet` method on `read`:

```
parquetFile = "/mnt/training/wikipedia/pageviews/pageviews_by_second.parquet"

(spark.read          # The DataFrameReader
 .parquet(parquetFile)  # Creates a DataFrame from Parquet after reading
 in the file
 .printSchema()        # Print the DataFrame's schema
)
```

A few important things to note are that we do not need to specify the schema - the column names and data types are stored in the parquet files. Only one job is required to **read** that schema from the parquet file's metadata. Unlike the CSV or JSON readers that have to load the entire file and then infer the schema, the parquet reader can "read" the schema very quickly because it's reading that schema from the metadata.

Read in the Parquet files with a schema

If you want to avoid the extra job entirely, we can, again, specify the schema even for parquet files:

WARNING: Providing a schema may avoid this one-time hit to determine the DataFrame's schema. However, if you specify the wrong schema, it will conflict with the true schema and will result in an analysis exception at runtime.

```
# Required for StructField, StringType, IntegerType, etc.
from pyspark.sql.types import *

parquetSchema = StructType(
 [
    StructField("timestamp", StringType(), False),
    StructField("site", StringType(), False),
    StructField("requests", IntegerType(), False)
]
)

(spark.read          # The DataFrameReader
 .schema(parquetSchema) # Use the specified schema
 .parquet(parquetFile)  # Creates a DataFrame from Parquet after reading
 in the file
 .printSchema()        # Print the DataFrame's schema
)
```

In most/many cases, people do not provide the schema for Parquet files because reading in the schema is such a cheap process.

Continue to the next step

After you've completed the notebook, return to this screen, and continue to the next step.

Read data stored in tables and views

In your Azure Databricks workspace, open the **03-Reading-and-writing-data-in-Azure-Databricks** folder that you imported within your user folder.

Open the **4.Reading Data - Tables and Views** notebook. Make sure you attach your cluster to the notebook before following the instructions and running the cells within.

Within the notebook, you will:

- Demonstrate how to pre-register data sources in Azure Databricks
- Introduce temporary views over files
- Read data from tables/views

Registering and reading tables in Azure Databricks

In this notebook, you start with registering a new table. So far we've seen purely programmatic methods for reading in data. Azure Databricks allows us to "register" the equivalent of "tables" so that they can be easily accessed by all users. It also allows us to specify configuration settings such as secret keys, tokens, username & passwords, etc. without exposing that information to all users.

There are several benefits to registering tables:

- Once setup, it never has to be done again
- It is available for any user on the platform (permissions permitting)
- Minimizes exposure of credentials
- No real overhead to reading the schema (no infer-schema)
- Easier to advertise available datasets to other users

Once you create a table, you can read it into a new DataFrame with one simple command:

```
pageviewsBySecondsExampleDF = spark.read.table("pageviews_by_second_example_tsv")
```

When reading from Databricks tables, no job is executed - the schema is stored in the table definition on Databricks. The data types are defined when you register the table. In the notebook, the file you upload to Azure Databricks is stored on the DBFS (Databricks File System). If you used JDBC, it would open the connection to the database and read it in. If you used an object store (like what is backing the DBFS), it would read the data from source.

The "registration" of the table simply makes future access, or access by multiple users easier. The users of the notebook cannot see username and passwords, secret keys, tokens, etc.

Temporary views

Tables that are loadable by the call `spark.read.table(..)` are also accessible through the SQL APIs. For example, we already used Databricks to expose **pageviews_by_second_example_tsv** as a table/view.

```
%sql  
select * from pageviews_by_second_example_tsv limit(5)
```

You can also take an existing `DataFrame` and register it as a view exposing it as a table to the SQL API.

If you recall from earlier, we have an instance called `parquetDF`. We can create a [temporary] view with this call:

```
# create a DataFrame from a parquet file  
parquetFile = "/mnt/training/wikipedia/pagecounts/staging_parquet_en_only_clean/"  
parquetDF = spark.read.parquet(parquetFile)  
  
# create a temporary view from the resulting DataFrame  
parquetDF.createOrReplaceTempView("parquet_table")
```

And now we can use the SQL API to reference that same `DataFrame` as the table **parquet_table**:

```
%sql  
select * from parquet_table order by requests desc limit(5)
```

Note #1: The method `createOrReplaceTempView(..)` is bound to the `SparkSession` meaning it will be discarded once the session ends.

Note #2: On the other hand, the method `createOrReplaceGlobalTempView(..)` is bound to the spark application.

Or to put that another way, you can use `createOrReplaceTempView(..)` in this notebook only. However, you can call `createOrReplaceGlobalTempView(..)` in this notebook and then access it from another.

Continue to the next step

After you've completed the notebook, return to this screen, and continue to the next step.

Write data

In your Azure Databricks workspace, open the **03-Reading-and-writing-data-in-Azure-Databricks** folder that you imported within your user folder.

Open the **5.Writing Data** notebook. Make sure you attach your cluster to the notebook before following the instructions and running the cells within.

Within the notebook, you will:

- Write data to a Parquet file
- Read the Parquet file back and display the results

Writing data

In this notebook, you load the original CSV data source that you used in an earlier notebook, then write the contents of that `DataFrame` to Parquet format:

```
fileName = userhome + "/pageviews_by_second.parquet"
print("Output location: " + fileName)

(csvDF.write                      # Our DataFrameWriter
 .option("compression", "snappy")  # One of none, snappy, gzip, and lzo
 .mode("overwrite")                # Replace existing files
 .parquet(fileName)               # Write DataFrame to Parquet files
)
```

Notice the `parquet` method on `write`. This specifies the Parquet format for the new file, although the original file you read was in CSV format.

Continue to the next step

After you've completed the notebook, return to this screen, and continue to the next step.

Exercises: Read and write data

In your Azure Databricks workspace, open the **03-Reading-and-writing-data-in-Azure-Databricks** folder that you imported within your user folder.

Open the **6.Reading Data - Lab** notebook. Make sure you attach your cluster to the notebook before following the instructions and running the cells within.

The goal of this exercise is to put into practice some of what you have learned about reading data with Apache Spark. The instructions are provided within the notebook, along with empty cells for you to do your work. At the bottom of the notebook are additional cells that will help verify that your work is accurate.

Note: You will find a corresponding notebook within the `Solutions` subfolder. This contains completed cells for the exercise. Refer to the notebook if you get stuck or simply want to see the solution.

After you've completed the notebook, return to this screen, and continue to the next step.

Knowledge check

Question 1

How do you list files in DBFS within a notebook?

- `ls /my-file-path.`
- `%fs dir /my-file-path.`
- `%fs ls /my-file-path.`

Question 2

How do you infer the data types and column names when you read a JSON file?

- `spark.read.option("inferSchema", "true").json(jsonFile).`
- `spark.read.inferSchema("true").json(jsonFile).`
- `spark.read.option("inferData", "true").json(jsonFile).`

Summary

In this module, you learned the basics about reading and writing data in Azure Databricks. You now know how to read CSV, JSON, and Parquet file formats, and how to write Parquet files to the Databricks file system (DBFS) with compression options. Though you only wrote the files in Parquet format, you can use the same `DataFrame.write` method to output to other formats. Finally, you put your knowledge to the test by completing an exercise that required you to read a random file that you had not yet seen.

Now that you have concluded this module, you should know:

1. Read data from CSV files into a Spark Dataframe
2. Provide a Schema when reading Data into a Spark Dataframe
3. Read data from JSON files into a Spark Dataframe
4. Read Data from parquet files into a Spark Dataframe
5. Create Tables and Views
6. Write data from a Spark Dataframe

Clean up

If you plan on completing other Azure Databricks modules, don't delete your Azure Databricks instance yet. You can use the same environment for the other modules.

Delete the Azure Databricks instance

1. Navigate to the Azure portal.
2. Navigate to the resource group that contains your Azure Databricks instance.
3. Select **Delete resource group**.
4. Type the name of the resource group in the confirmation text box.
5. Select **Delete**.

Work with DataFrames in Azure Databricks

Introduction

Suppose you're working for a data analytics startup that's now expanding along with its increasing customer base. You receive customer data from multiple sources in different raw formats. To efficiently handle huge amounts of customer data, your company has decided to invest in Azure Databricks. Your team is responsible for analyzing how Databricks supports day-to-day data-handling functions, such as reads, writes, and queries. You want to move beyond simple data-handling functions and learn ways to simplify data exploration and perform data transformations. The data exploration and transformation tasks help your team conduct advanced analytics and machine learning in later stages of the data pipeline.

Learning objectives

In this module, you'll:

1. Use the `count()` method to count rows in a DataFrame
2. Use the `display()` function to display a DataFrame in the Notebook
3. Cache a DataFrame for quicker operations if the data is needed a second time
4. Use the `limit` function to display a small set of rows from a larger DataFrame
5. Use `select()` to select a subset of columns from a DataFrame
6. Use `distinct()` and `dropDuplicates` to remove duplicate data
7. Use `drop()` to remove columns from a DataFrame

Prerequisites

- An Azure subscription. If you don't have an Azure subscription, create a [free account¹⁴](#).

Describe a DataFrame

In this unit, you need to complete the exercises within a Databricks Notebook. To begin, you need to have access to an Azure Databricks workspace. If you do not have a workspace available, follow the instructions below. Otherwise, you can skip to the bottom of the page to Clone the Databricks archive.

Unit Pre-requisites

Microsoft Azure Account: You will need a valid and active Azure account for the Azure labs. If you do not have one, you can sign up for a [free trial¹⁵](#)

- If you are a Visual Studio Active Subscriber, you are entitled to azure credits per month. You can refer to this [link¹⁶](#) to find out more including how to activate and start using your monthly Azure credit.
- If you are not a Visual Studio Subscriber, you can sign up for the FREE [Visual Studio Dev Essentials¹⁷](#) program to create Azure free account.

¹⁴ <https://azure.microsoft.com/free>

¹⁵ <https://azure.microsoft.com/free/>

¹⁶ <https://azure.microsoft.com/pricing/member-offers/msdn-benefits-details/>

¹⁷ <https://www.visualstudio.com/dev-essentials/>

Create the required resources

To complete this lab, you will need to deploy an Azure Databricks workspace in your Azure subscription.

Deploy an Azure Databricks workspace

1. Click the following button to open the ARM template in the Azure portal.
[Deploy Databricks from the ARM Template¹⁸](#)
2. Provide the required values to create your Azure Databricks workspace:
 - **Subscription:** Choose the Azure Subscription in which to deploy the workspace.
 - **Resource Group:** Leave at Create new and provide a name for the new resource group.
 - **Location:** Select a location near you for deployment. For the list of regions supported by Azure Databricks, see [Azure services available by region¹⁹](#).
 - **Workspace Name:** Provide a name for your workspace.
 - **Pricing Tier:** Ensure premium is selected.
3. Accept the terms and conditions.
4. Select Purchase.
5. The workspace creation takes a few minutes. During workspace creation, the portal displays the Submitting deployment for Azure Databricks tile on the right side. You may need to scroll right on your dashboard to see the tile. There is also a progress bar displayed near the top of the screen. You can watch either area for progress.

Create a cluster

1. When your Azure Databricks workspace creation is complete, select the link to go to the resource.
2. Select **Launch Workspace** to open your Databricks workspace in a new tab.
3. In the left-hand menu of your Databricks workspace, select **Clusters**.
4. Select **Create Cluster** to add a new cluster.

¹⁸ <https://portal.azure.com/#create/Microsoft.Template/uri/https%3A%2F%2Fraw.githubusercontent.com%2FAzure%2Fazure-quickstart-templates%2Fmaster%2F101-databricks-workspace%2Fazuredeploy.json>

¹⁹ <https://azure.microsoft.com/regions/services/>

Create Cluster

New Cluster

[Cancel](#)

[Create Cluster](#)

2-8 Workers: 28.0-112.0 GB Memory, 8-32 Cores
1 Driver: 14.0 GB Memory, 4 Cores, 0.75 DBU

Cluster Name

Test Cluster



Cluster Mode

Standard

Pool

None

Databricks Runtime Version

[Learn more](#)

Runtime: 6.4 (Scala 2.11, Spark 2.4.5)

New This Runtime version supports only Python 3.

Autopilot Options

Enable autoscaling

Terminate after minutes of inactivity

Worker Type

Standard_DS3_v2

14.0 GB Memory, 4 Cores, 0.75 DBU

Min Workers

2

Max Workers

8

Driver Type

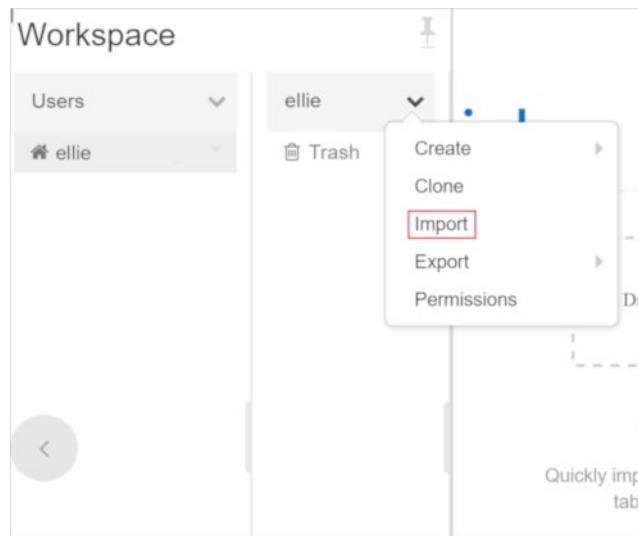
Same as worker

14.0 GB Memory, 4 Cores, 0.75 DBU

- Enter a name for your cluster. Use your name or initials to easily differentiate your cluster from your coworkers.
- Select the **Databricks RuntimeVersion**. We recommend the latest runtime and **Scala 2.12**.
- Select the default values for the cluster configuration.
- Select **Create Cluster**.

Clone the Databricks archive

- If you do not currently have your Azure Databricks workspace open: in the Azure portal, navigate to your deployed Azure Databricks workspace and select **Launch Workspace**.
- In the left pane, select **Workspace > Users**, and select your username (the entry with the house icon).
- In the pane that appears, select the arrow next to your name, and select **Import**.



4. In the **Import Notebooks** dialog box, select the URL and paste in the following URL:

<https://github.com/solliancenet/microsoft-learning-paths-databricks-notebooks/blob/master/data-engineering/DBC/04-Working-With-Dataframesdbc?raw=true>

1. Select **Import**.
2. Select the **04-Working-With-Dataframes** folder that appears.

Complete the following notebook

Open the **1.Describe-a-dataframe** notebook. Make sure you attach your cluster to the notebook before following the instructions and running the cells within.

Within the notebook, you will:

- Develop familiarity with the `DataFrame` APIs
- Learn the classes...
 - `SparkSession`
 - `DataFrame` (aka `Dataset[Row]`)
- Learn the action...
 - `count()`

Create a DataFrame

In this notebook, you start by creating a new `DataFrame`. You can read the Parquet files into a `DataFrame`. To do this, you start with the object `spark`, an instance of `SparkSession` and the entry point to Spark 2.0 applications. From there you can access the `read` object which gives you an instance of `DataFrameReader`:

```
parquetDir = source + "/wikipedia/pagecounts/staging_parquet_en_only_clean/"

pagecountsEnAllDF = (spark # Our SparkSession & Entry Point
    .read # Our DataFrameReader
    .parquet(parquetDir) # Returns an instance of DataFrame
)
print(pagecountsEnAllDF) # Python hack to see the data type
```

The `count()` method triggers a job to process a request to count the number of rows in a dataset, and returns a value.

You can now count all records in the `DataFrame` like this:

```
total = pagecountsEnAllDF.count()
```

Continue to the next step

After you've completed the notebook, return to this screen, and continue to the next step.

Use common DataFrame methods

In your Azure Databricks workspace, open the **04-Working-With-Dataframes** folder that you imported within your user folder.

Open the **2.Use-common-dataframe-methods** notebook. Make sure you attach your cluster to the notebook before following the instructions and running the cells within.

Within the notebook, you will:

- Develop familiarity with the `DataFrame` APIs
- Use common `DataFrame` methods for performance
- Explore the Spark API documentation

cache() & persist()

In this notebook, you start by creating the same `DataFrame` you created in the previous notebook:

```
parquetDir = source + "/wikipedia/pagecounts/staging_parquet_en_only_clean/"

pagecountsEnAllDF = (spark # Our SparkSession & Entry Point
    .read # Our DataFrameReader
    .parquet(parquetDir) # Returns an instance of DataFrame
)
print(pagecountsEnAllDF) # Python hack to see the data type
```

There are around 2 million rows in the `DataFrame`, which we discover by executing `pagecountsEnAllDF.count()`.

With a `DataFrame` of this size, we want to optimize the processing where possible.

The ability to cache data is one technique for achieving better performance with Apache Spark. This is because every action requires Spark to read the data from its source (Azure Blob, Amazon S3, HDFS, etc.), but caching moves that data into the memory of the local executor for “instant” access.

`cache()` is just an alias for `persist()`. We can use `cache` in the following way:

```
(pagecountsEnAllDF  
    .cache()          # Mark the DataFrame as cached  
    .count()          # Materialize the cache  
)
```

When you rerun the `count` command, it should take significantly less time (`pagecountsEnAllDF.count()`).

When caching data, you are placing it on the workers of the cluster. Caching takes resources. Before moving a notebook into production, please check and verify that you are appropriately using `cache`.

Continue to the next step

After you've completed the notebook, return to this screen, and continue to the next step.

Use the display function

In your Azure Databricks workspace, open the **04-Working-With-Dataframes** folder that you imported within your user folder.

Open the **3.Display-function** notebook. Make sure you attach your cluster to the notebook before following the instructions and running the cells within.

Within the notebook, you will:

- Learn the transformations...

- `limit(..)`
 - `select(..)`
 - `drop(..)`
 - `distinct()`
 - `dropDuplicates(..)`
 - Learn the actions...
- `show(..)`
 - `display(..)`

show(...)

In this notebook, you start by creating the same `DataFrame` you created in the previous notebook:

```
parquetDir = source + "/wikipedia/pagecounts/staging_parquet_en_only_clean/"  
  
pagecountsEnAllDF = (spark # Our SparkSession & Entry Point
```

```
.read          # Our DataFrameReader
.parquet(parquetDir) # Returns an instance of DataFrame
)
print(pagecountsEnAllDF) # Python hack to see the data type
```

What we want to look for next is a function that will allow us to print the data to the console.

In the API docs for `DataFrame/Dataset` find the docs for the `show(...)` command(s).

In the case of Python, we have one method with two optional parameters.

In the case of Scala, we have several overloaded methods.

In either case, the `show(...)` method effectively has two optional parameters:

- **n**: The number of records to print to the console, the default being 20.
- **truncate**: If true, columns wider than 20 characters will be truncated, where the default is true.

Let's take a look at the data in our `DataFrame` with the `show()` command:

```
pagecountsEnAllDF.show()
```

Note: The function `show(...)` is an **action** which triggers a job.

display(..)

The `show(...)` command is part of the core Spark API and simply prints the results to the console.

Our notebooks have a slightly more elegant alternative.

Instead of calling `show(...)` on an existing `DataFrame` we can instead pass our `DataFrame` to the `display(..)` command:

```
display(pagecountsEnAllDF)
```

show(..) vs display(..)

- `show(...)` is part of core spark - `display(..)` is specific to our notebooks.
- `show(...)` is ugly - `display(..)` is pretty.
- `show(...)` has parameters for truncating both columns and rows - `display(..)` does not.
- `show(...)` is a function of the `DataFrame/Dataset` class - `display(..)` works with a number of different objects.
- `display(..)` is more powerful - with it, you can...
 - Download the results as CSV
 - Render line charts, bar chart & other graphs, maps and more.
 - See up to 1000 records at a time.

For the most part, the difference between the two is going to come down to preference.

Like `DataFrame.show(..)`, `display(..)` is an **action** which triggers a job.

limit(..)

Both `show(..)` and `display(..)` are **actions** that trigger jobs (though in slightly different ways).

If you recall, `show(..)` has a parameter to control how many records are printed but, `display(..)` does not. We can address that difference with our first transformation, `limit(..)`.

If you look at the API docs, `limit(..)` is described like this:

Returns a new Dataset by taking the first n rows...

`show(..)`, like many actions, does not return anything. On the other hand, transformations like `limit(..)` return a **new DataFrame**:

```
limitedDF = pagecountsEnAllDF.limit(5) # "limit" the number of records to  
the first 5
```

Notice how “nothing” happened - that is no job was triggered. This is because we are simply defining the second step in our transformations:

- Read in the parquet file (represented by **pagecountsEnAllDF**).
- Limit those records to just the first 5 (represented by **limitedDF**).

It is not until we induce an action that a job is triggered and the data is processed. We can induce a job by calling either the `show(..)` or the `display(..)` actions:

```
limitedDF.show(100, False) #show up to 100 records and don't truncate the  
columns
```

select(..)

Let's say, for the sake of argument, that we don't want to look at all the data. For example, it was asserted in the query results above that **bytes_served** had nothing but zeros in it and consequently is of no value to us.

If that is the case, we can disregard it by selecting only the three columns that we want:

```
# Transform the data by selecting only three columns  
onlyThreeDF = (pagecountsEnAllDF  
    .select("project", "article", "requests") # Our 2nd transformation (4 >>  
3 columns)  
)
```

Again, notice how the call to `select(..)` does not trigger a job. That is because `select(..)` is a transformation. It's just one more step in a long list of transformations.

Let's go ahead and invoke the action `show(..)` and take a look at the result:

```
# And lastly, show the first five records which should exclude the bytes_  
served column.  
onlyThreeDF.show(5, False)
```

The output should be:

project	article	requests

```
+-----+-----+
| len | !?Revolution!? | 1 |
| len | !Ay,_caramba! | 1 |
| len | !DOCTYPE       | 1 |
| len | !Gã!nge_language| 1 |
| len | !Hukwe_language | 1 |
+-----+-----+
only showing top 5 rows
```

The `select(..)` command is one of the most powerful and most commonly used transformations.

If you look at the API docs, `select(..)` is described like this:

Returns a new Dataset by computing the given Column expression for each element.

The “Column expression” referred to there is where the true power of this operation shows up. Again, we will go deeper on these later. Just like `limit(..)`, `select(..)` does not trigger a job, returns a new `DataFrame`, and simply defines the next transformation in a sequence of transformations.

drop(..)

As a quick side note, you will quickly discover there are a lot of ways to accomplish the same task. Take the transformation `drop(..)` for example - instead of selecting everything we wanted, `drop(..)` allows us to specify the columns we don't want.

If you look at the API docs, `drop(..)` is described like this:

Returns a new Dataset with a column dropped.

And we can see that we can produce the same result as the last exercise this way:

```
# Transform the data by selecting only three columns
droppedDF = (pagecountsEnAllDF
    .drop("bytes_served") # Our second transformation after the initial read
    (4 columns down to 3)
)
```

Again, `drop(..)` is just one more transformation - that is no job is triggered. Just as we did above with `select`, we can invoke a job and apply the transformation with `show`:

```
# And lastly, show the first five records which should exclude the bytes_
served column.
droppedDF.show(5, False)
```

distinct() & dropDuplicates()

These two transformations do the same thing. In fact, they are aliases for one another. You can see this by looking at the source code for these two methods:

- `def distinct(): Dataset[T] = dropDuplicates()`
- See **Dataset.scala**²⁰

²⁰ <https://github.com/apache/spark/blob/master/sql/core/src/main/scala/org/apache/spark/sql/Dataset.scala>

The difference between them has everything to do with the programmer and their perspective.

- The name **distinct** will resonate with developers, analysts, and DB admins with a background in SQL.
- The name **dropDuplicates** will resonate with developers that have a background or experience in functional programming.

As you become more familiar with the various APIs, you will see this pattern reassert itself. The designers of the API are trying to make the API as approachable as possible for multiple target audiences.

If you look at the API docs, both `distinct(..)` and `dropDuplicates(..)` are described like this:

Returns a new Dataset that contains only the unique rows from this Dataset....

With this transformation, we can now tackle our first business question:

How many different English Wikimedia projects saw traffic during that hour?

If you recall, our original DataFrame has this schema:

```
root
|-- project: string (nullable = true)
|-- article: string (nullable = true)
|-- requests: integer (nullable = true)
|-- bytes_served: long (nullable = true)
```

The transformation `distinct()` is applied to the row as a whole - data in the **project**, **article** and **requests** column will effect this evaluation. To get the distinct list of projects, and only projects, we need to reduce the number of columns to just the one column, **project**.

We can do this with the `select(..)` transformation and then we can introduce the `distinct()` transformation:

```
distinctDF = (pagecountsEnAllDF      # Our original DataFrame from spark.
              .read.parquet(..)
              .select("project")           # Drop all columns except the "project"
              .column
              .distinct())                # Reduce the set of all records to just
              the distinct column.
            )
```

Just to reinforce, we have three transformations:

1. Read the data (now represented by `pagecountsEnAllDF`)
2. Select just the one column
3. Reduce the records to a distinct set

No job is triggered until we perform an action like `show(..)`:

```
# There will not be more than 100 projects
distinctDF.show(100, False)
```

dropDuplicates(columns...)

The method `dropDuplicates(..)` has a second variant that accepts one or more columns. The distinction is not performed across the entire record unlike `distinct()` or even `dropDuplicates()`.

The distinction is based only on the specified columns. This allows us to keep all the original columns in our DataFrame.

Recap

Our code is spread out over many cells which can make this a little hard to follow. Let's take a look at the same code in a single cell:

```
pagecountsEnAllDF = (spark
    .read
    .parquet(parquetDir)
)
(pagecountsEnAllDF
    .cache()
    .count()
)
distinctDF = (pagecountsEnAllDF
    .select("project")
    .distinct()
)
total = distinctDF.count()
```

DataFrames vs SQL & temporary views

The DataFrames API is built upon an SQL engine. As such, we can "convert" a DataFrame into a temporary view (or table) and then use it in "standard" SQL.

Let's start by creating a temporary view from a previous DataFrame:

```
pagecountsEnAllDF.createOrReplaceTempView("pagecounts")
```

Now that we have a temporary view (or table) we can start expressing our queries and transformations in SQL:

```
%sql
SELECT * FROM pagecounts
```

And we can just as easily express in SQL the distinct list of projects, and just because we can, we'll sort that list:

```
%sql
SELECT DISTINCT project FROM pagecounts ORDER BY project
```

And converting from SQL back to a DataFrame is just as easy:

```
tableDF = spark.sql("SELECT DISTINCT project FROM pagecounts ORDER BY
project")
```

```
display(tableDF)
```

Continue to the next step

After you've completed the notebook, return to this screen, and continue to the next step.

Exercise: Distinct articles

In your Azure Databricks workspace, open the **04-Working-With-Dataframes** folder that you imported within your user folder.

Open the **4.Exercise: Distinct Articles** notebook. Make sure you attach your cluster to the notebook before following the instructions and running the cells within.

In this exercise, you read Parquet files, apply necessary transformations, perform a total count of records, then verify that all the data was correctly loaded. As a bonus, try defining a schema that matches the data and update the read operation to use the schema.

Note: You will find a corresponding notebook within the `Solutions` subfolder. This contains completed cells for the exercise. Refer to the notebook if you get stuck or simply want to see the solution.

After you've completed the notebook, return to this screen, and continue to the next step.

Knowledge check

Question 1

Which DataFrame method do you use to create a temporary view?

- `createTempView()`
- `createTempViewDF()`
- `createOrReplaceTempView()`

Question 2

How do you create a DataFrame object?

- Introduce a variable name and equate it to something like `myDataFrameDF =`
- Use the `createDataFrame()` function
- Use the `DF.create()` syntax

Question 3

How do you cache data into the memory of the local executor for instant access?

- `.save().inMemory()`
- `.inMemory().save()`
- `.cache()`

Question 4

What is the Python syntax for defining a DataFrame in Spark from an existing Parquet file in DBFS?

- IPGeocodeDF = parquet.read("dbfs:/mnt/training/ip-geocode.parquet")
- IPGeocodeDF = spark.read.parquet("dbfs:/mnt/training/ip-geocode.parquet")
- IPGeocodeDF = spark.parquet.read("dbfs:/mnt/training/ip-geocode.parquet")

Summary

DataFrames in Azure Databricks help you move beyond simple data-handling functions by simplifying data exploration and perform data transformations. You can access numerous types of data sources and apply these same transformations, aggregates, and caching functions.

Cleanup

If you plan to complete other Azure Databricks modules, don't delete your Azure Databricks instance yet. You can use the same environment for the other modules.

Delete the Azure Databricks instance

1. Sign in to the [Azure portal²¹](#).
2. In the left pane, select **Resource groups**, and then find the resource group that contains your Azure Databricks instance.
3. Select the resource group, and either right-click the row or use the **ellipsis (...)** button to open the context menu.
4. Select **Delete resource group**.
5. Enter the name of the resource group, and then select **Delete**.

²¹ <https://portal.azure.com?azure-portal=true>

Work with DataFrames advanced methods in Azure Databricks

Introduction

Suppose you're working for a data analytics company that has expanded to the point of transitioning from a small startup to an established business with a great track record. You have been using `DataFrames` extensively in Azure Databricks to apply transformations, read and write data, and perform advanced data analysis. You have recently added a new data source, which requires you to rename columns to conform to your existing data warehouse. Plus, you need to perform date/time conversions for compatibility reasons. On top of these changes, you apply a series of aggregates over your data to build new reports.

Learning objectives

In this module, you'll:

- Manipulate date and time values in Azure Databricks
- Rename columns in Azure Databricks
- Aggregate data in Azure Databricks `DataFrames`

Prerequisites

- An Azure subscription. If you don't have an Azure subscription, create a [free account²²](#).

Perform date and time manipulation

In this unit, you need to complete the exercises within a Databricks Notebook. To begin, you need to have access to an Azure Databricks workspace. If you do not have a workspace available, follow the instructions below. Otherwise, you can skip to the bottom of the page to Clone the Databricks archive.

Unit Pre-requisites

Microsoft Azure Account: You will need a valid and active Azure account for the Azure labs. If you do not have one, you can sign up for a [free trial²³](#)

- If you are a Visual Studio Active Subscriber, you are entitled to Azure credits per month. You can refer to this [link²⁴](#) to find out more including how to activate and start using your monthly Azure credit.
- If you are not a Visual Studio Subscriber, you can sign up for the FREE [Visual Studio Dev Essentials²⁵](#) program to create Azure free account.

Create the required resources

To complete this lab, you will need to deploy an Azure Databricks workspace in your Azure subscription.

²² <https://azure.microsoft.com/free>

²³ <https://azure.microsoft.com/free/>

²⁴ <https://azure.microsoft.com/pricing/member-offers/msdn-benefits-details/>

²⁵ <https://www.visualstudio.com/dev-essentials/>

Deploy an Azure Databricks workspace

1. Click the following button to open the Azure Resource Manager template in the Azure portal.
Deploy Databricks from the Azure Resource Manager Template²⁶
2. Provide the required values to create your Azure Databricks workspace:
 - **Subscription:** Choose the Azure Subscription in which to deploy the workspace.
 - **Resource Group:** Leave at Create new and provide a name for the new resource group.
 - **Location:** Select a location near you for deployment. For the list of regions supported by Azure Databricks, see **Azure services available by region²⁷**.
 - **Workspace Name:** Provide a name for your workspace.
 - **Pricing Tier:** Ensure premium is selected.
3. Accept the terms and conditions.
4. Select Purchase.
5. The workspace creation takes a few minutes. During workspace creation, the portal displays the Submitting deployment for Azure Databricks tile on the right side. You may need to scroll right on your dashboard to see the tile. There is also a progress bar displayed near the top of the screen. You can watch either area for progress.

Create a cluster

1. When your Azure Databricks workspace creation is complete, select the link to go to the resource.
2. Select **Launch Workspace** to open your Databricks workspace in a new tab.
3. In the left-hand menu of your Databricks workspace, select **Clusters**.
4. Select **Create Cluster** to add a new cluster.

²⁶ <https://portal.azure.com/#create/Microsoft.Template/uri/https%3A%2F%2Fraw.githubusercontent.com%2FAzure%2Fazure-quickstart-templates%2Fmaster%2F101-databricks-workspace%2Fazuredeploy.json>

²⁷ <https://azure.microsoft.com/regions/services/>

Create Cluster

New Cluster | [Cancel](#) | [Create Cluster](#) | 2-8 Workers: 28.0-112.0 GB Memory, 8-32 Cores, 0.75 DBU
1 Driver: 14.0 GB Memory, 4 Cores, 0.75 DBU

Cluster Name: Test Cluster

Cluster Mode: Standard

Pool: None

Databricks Runtime Version: [Learn more](#)
Runtime: 6.4 (Scala 2.11, Spark 2.4.5)

New This Runtime version supports only Python 3.

Autopilot Options:
 Enable autoscaling
 Terminate after 120 minutes of inactivity

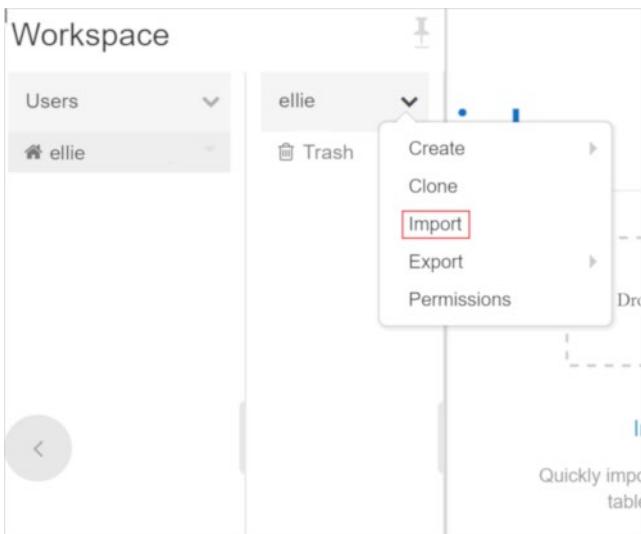
Worker Type	Min Workers	Max Workers	
Standard_DS3_v2	14.0 GB Memory, 4 Cores, 0.75 DBU	2	8

Driver Type: Same as worker

5. Enter a name for your cluster. Use your name or initials to easily differentiate your cluster from your coworkers.
6. Select the **Databricks RuntimeVersion**. We recommend the latest runtime and **Scala 2.12**.
7. Select the default values for the cluster configuration.
8. Select **Create Cluster**.

Clone the Databricks archive

1. If you do not currently have your Azure Databricks workspace open: in the Azure portal, navigate to your deployed Azure Databricks workspace and select **Launch Workspace**.
2. In the left pane, select **Workspace > Users**, and select your username (the entry with the house icon).
3. In the pane that appears, select the arrow next to your name, and select **Import**.



4. In the **Import Notebooks** dialog box, select the URL and paste in the following URL:

```
https://github.com/solliancenet/microsoft-learning-paths-databricks-notebooks/blob/master/data-engineering/DBC/07-Dataframe-Advanced-Methodsdbc?raw=true
```

5. Select **Import**.
6. Select the **07-Dataframe-Advanced-Methods** folder that appears.

Complete the following notebook

Open the **1.DateTime-Manipulation** notebook. Make sure you attach your cluster to the notebook before following the instructions and running the cells within.

Within the notebook, you will:

- Explore more of the ...sql.functions operations
 - Date & time functions

withColumnRenamed(..), withColumn(..), select(..)

In this notebook, you will load Parquet files that contain Wikipedia page views by second. The resulting DataFrame has a **column named timestamp** and the **datatype will also be timestamp**. Apache Spark makes this easy to fix by renaming that column using alias:

```
(initialDF
    .select( col("timestamp").alias("capturedAt"), col("site"), col("requests") )
    .printSchema()
)
```

Or by using `withColumnRenamed`:

```
(initialDF
    .withColumnRenamed("timestamp", "capturedAt")
    .printSchema()
)
```

unix_timestamp(..) & cast(..)

Next, you convert the **string** type for the date/time field to a **timestamp**.

For this we will be looking at more of the functions in the `functions` package:

- `pyspark.sql.functions` in the case of Python
- `org.apache.spark.sql.functions` in the case of Scala & Java

The first function is `unix_timestamp(..)`

If you look at the API docs, `unix_timestamp(..)` is described like this:

Convert time string with given pattern (see [SimpleDateFormat²⁸](#)) to Unix time stamp (in seconds), return null if fail.

`SimpleDateFormat` is part of the Java API and provides support for parsing and formatting date and time values.

In order to know what format the data is in, let's take a look at the first row. Comparing that value with the patterns express in the docs for the `SimpleDateFormat` class, we can come up with a format:

yyyy-MM-dd HH:mm:ss

```
tempA = (initialDF
    .withColumnRenamed("timestamp", "capturedAt")
    .select( col("*"), unix_timestamp( col("capturedAt"), "yyyy-MM-dd HH:mm:ss" ) )
)
display(tempA)
```

The output of this command is:

capturedAt	site	requests	unix_timestamp(capturedAt, yyyy-MM-dd HH:mm:ss)
2015-04-07T18:57:19	desktop	3162	null
2015-04-07T00:42:37	desktop	2546	null
2015-04-08T14:46:13	desktop	3254	null
2015-04-18T14:25:35	mobile	1348	null
2015-04-19T10:18:58	mobile	1014	null

There is a bug with this code.

A few things happened:

1. We ended up with a new column - that's OK for now

²⁸ <http://docs.oracle.com/javase/tutorial/i18n/format/simpleDateFormat.html>

2. The new column has a really funky name - based upon the name of the function we called and its parameters.
3. The data type is now a long.
 - This value is the Java Epoch
 - The number of seconds since 1970-01-01T00:00:00Z

We can now take that epoch value and use the `Column.cast(..)` method to convert it to a **timestamp**:

```
tempB = (initialDF
    .withColumnRenamed("timestamp", "capturedAt")
    .select( col("*"), unix_timestamp( col("capturedAt") ), "yyyy-MM-dd'T'HH:mm:ss").cast("timestamp") )
)
display(tempB)
```

The output of this new command is:

capturedAt	site	requests	CAST(unix_timestamp(-capturedAt, yyyy-MM-dd'T'HH:mm:ss) AS TIMESTAMP)
2015-04-07T18:57:19	desktop	3162	2015-04-20T22:40:24.000+0000
2015-04-07T00:42:37	desktop	2546	2015-04-20T22:32:02.000+0000
2015-04-08T14:46:13	desktop	3254	2015-04-20T22:24:25.000+0000
2015-04-18T14:25:35	mobile	1348	2015-04-20T21:45:21.000+0000
2015-04-19T10:18:58	mobile	1014	2015-04-20T21:11:35.000+0000

There are several options to deal with the strange column name. You will see these options in the notebook, but the cleanest option uses the `withColumn(..)` to rename the column (first param) and accept as a second parameter the expression(s) we need for our transformation:

```
pageviewsDF = (initialDF
    .withColumnRenamed("timestamp", "capturedAt")
    .withColumn("capturedAt", unix_timestamp( col("capturedAt") ), "yyyy-MM-dd'T'HH:mm:ss").cast("timestamp") )
)
display(pageviewsDF)
```

The new output is:

capturedAt	site	requests
2015-04-07T18:57:19.000+0000	desktop	3162
2015-04-07T00:42:37.000+0000	desktop	2546

capturedAt	site	requests
2015-04-08T14:46:13.000+0000	desktop	3254
2015-04-18T14:25:35.000+0000	mobile	1348
2015-04-19T10:18:58.000+0000	mobile	1014

year(..), month(..), dayofyear(..)

Let's take a look at some of the other date & time functions. With that, we can answer a simple question: When was this data captured?

We can start specifically with the year:

```
display(  
    pageviewsDF  
        .select( year( col("capturedAt") ) ) # Every record converted to a  
        single column - the year captured  
        .distinct() # Reduce all years to the list of  
        distinct years  
)
```

The result is 2015.

Now let's take a look at in which months was this data captured:

```
display(  
    pageviewsDF  
        .select( month( col("capturedAt") ) ) # Every record converted to a  
        single column - the month captured  
        .distinct() # Reduce all months to the list of  
        distinct years  
)
```

This outputs 3 and 4 as the months in which the data was captured.

We can combine both as a single cell:

```
(pageviewsDF  
    .select( month(col("capturedAt")).alias("month"), year(col("capture-  
    dAt")).alias("year"))  
    .distinct()  
    .show()  
)
```

This outputs:

```
+----+----+  
|month|year|  
+----+----+  
|     3|2015|  
|     4|2015|  
+----+----+
```

It's pretty easy to see that the data was captured during March & April of 2015.

Continue to the next step

After you've completed the notebook, return to this screen, and continue to the next step.

Use aggregate functions

In your Azure Databricks workspace, open the **07-Dataframe-Advanced-Methods** folder that you imported within your user folder.

Open the **2.Use-Aggregate-Functions** notebook. Make sure you attach your cluster to the notebook before following the instructions and running the cells within.

Within the notebook, you will learn various aggregate functions.

groupBy()

Aggregating data is one of the more common tasks when working with big data.

- How many customers are over 65?
- What is the ratio of men to women?
- Group all emails by their sender.

The function `groupBy()` is one tool that we can use for this purpose.

If you look at the API docs, `groupBy(...)` is described like this:

Groups the Dataset using the specified columns, so that we can run aggregation on them.

This function is a **wide** transformation - it will produce a shuffle and conclude a stage boundary. Unlike all of the other transformations we've seen so far, this transformation does not return a `DataFrame`.

- In Scala it returns `RelationalGroupedDataset`
- In Python it returns `GroupedData`

This is because the call `groupBy(...)` is only 1/2 of the transformation. To see the other half, we need to take a look at its return type, `RelationalGroupedDataset`.

RelationalGroupedDataset

If we take a look at the API docs for `RelationalGroupedDataset`, we can see that it supports the following aggregations:

Method	Description
<code>avg(...)</code>	Compute the mean value for each numeric columns for each group.
<code>count(...)</code>	Count the number of rows for each group.
<code>sum(...)</code>	Compute the sum for each numeric columns for each group.
<code>min(...)</code>	Compute the min value for each numeric column for each group.
<code>max(...)</code>	Compute the max value for each numeric columns for each group.

Method	Description
mean(...)	Compute the average value for each numeric columns for each group.
agg(...)	Compute aggregates by specifying a series of aggregate columns.
pivot(...)	Pivots a column of the current DataFrame and perform the specified aggregation.

With the exception of `pivot(...)`, each of these functions return our new DataFrame.

Together, `groupBy(...)` and `RelationalGroupedDataset` (or `GroupedData` in Python) give us what we need to answer some basic questions. For Example, how many more requests did the desktop site receive than the mobile site receive?

For this all we need to do is group all records by `site` and then sum all the requests.

```
display(
    pageviewsDF
        .groupBy( col("site") )
        .sum()
)
```

This outputs:

site	sum(requests)
desktop	8737180972
mobile	4605797962

Notice above that we didn't actually specify which column we were summing. In this case you will actually receive a total for all numerical values. There is a performance catch to that - if you have 2, 5, 10 columns, then they will all be summed and you may only need one.

You can first reduce my columns to those that you wanted, or your can simply specify which column(s) to sum up (while renaming the sum column):

```
display(
    pageviewsDF
        .groupBy( col("site") )
        .sum("requests")
        .withColumnRenamed("sum(requests)", "totalRequests")
)
```

This outputs:

site	totalRequests
desktop	8737180972
mobile	4605797962

How about the total number of requests per site, or mobile vs desktop?

```
display(
    pageviewsDF
        .groupBy( col("site") )
        .count()
```

```
)
```

This outputs:

site	count
desktop	3600000
mobile	3600000

sum(), count(), avg(), min(), max()

The `groupBy(...)` operation is not our only option for aggregating.

The `...sql.functions` package actually defines a large number of aggregate functions:

- `org.apache.spark.sql.functions` in the case of Scala & Java
- `pyspark.sql.functions` in the case of Python

Let's take a look at our last two examples. We saw the count of records and the sum of records. Let's take do this a slightly different way. This time with the `...sql.functions` operations, with some formatting and other aggregate functions included:

```
(pageviewsDF
    .filter("site = 'mobile'")
    .select(
        format_number(sum(col("requests")), 0).alias("sum"),
        format_number(count(col("requests")), 0).alias("count"),
        format_number(avg(col("requests")), 2).alias("avg"),
        format_number(min(col("requests")), 0).alias("min"),
        format_number(max(col("requests")), 0).alias("max")
    )
    .show()
)

(pageviewsDF
    .filter("site = 'desktop'")
    .select(
        format_number(sum(col("requests")), 0),
        format_number(count(col("requests")), 0),
        format_number(avg(col("requests")), 2),
        format_number(min(col("requests")), 0),
        format_number(max(col("requests")), 0)
    )
    .show()
)
```

This outputs:

sum	count	avg	min	max
4,605,797,962	3,600,000	1,279.39	645	3,292

```
+-----+-----+
-----+-----+
-----+
| format_number(sum(requests), 0) | format_number(count(requests), 0) | format_
| format_number(avg(requests), 2) | format_number(min(requests), 0) | format_number(max-
| requests), 0) |
+-----+-----+
-----+-----+
-----+
|           8,737,180,972 |                      3,600,000 |
| 2,426.99 |                  1,312 |                      5,695 |
+-----+-----+
-----+-----+
-----+
```

Continue to the next step

After you've completed the notebook, return to this screen, and continue to the next step.

Exercise: Deduplication of data

In your Azure Databricks workspace, open the **07-Dataframe-Advanced-Methods** folder that you imported within your user folder.

Open the **3.Exercise-Deduplication-of-Data** notebook. Make sure you attach your cluster to the notebook before following the instructions and running the cells within.

The goal of this exercise is to put into practice some of what you have learned about using DataFrames, including renaming columns. The instructions are provided within the notebook, along with empty cells for you to do your work. At the bottom of the notebook are additional cells that will help verify that your work is accurate.

Note: You will find a corresponding notebook within the `Solutions` subfolder. This contains completed cells for the exercise. Refer to the notebook if you get stuck or simply want to see the solution.

After you've completed the notebook, return to this screen, and continue to the next step.

Knowledge check

Question 1

Which method for renaming a DataFrame's column is incorrect?

- `df.select(col("timestamp").alias("dateCaptured"))`
- `df.alias("timestamp", "dateCaptured")`
- `df.toDF("dateCaptured")`

Question 2

You need to find the average of sales transactions by storefront. Which of the following aggregates would you use?

- df.select(col("storefront")).avg("completedTransactions")
- df.groupBy(col("storefront")).avg(col("completedTransactions"))
- df.groupBy(col("storefront")).avg("completedTransactions")

Summary

In this module, you built upon your knowledge of DataFrames in Azure Databricks by implementing various aggregate functions as well as date and time functions.

Now that you have concluded this module, you should:

1. Know how to manipulate date and time values in Azure Databricks
2. Know how to rename columns in Azure Databricks
3. Know how to aggregate data in Azure Databricks DataFrames

Clean up

If you plan on completing other Azure Databricks modules, don't delete your Azure Databricks instance yet. You can use the same environment for the other modules.

Delete the Azure Databricks instance

1. Navigate to the Azure portal.
2. Navigate to the resource group that contains your Azure Databricks instance.
3. Select **Delete resource group**.
4. Type the name of the resource group in the confirmation text box.
5. Select **Delete**.

Module Lab information

Lab 3 - Data Exploration and Transformation in Azure Databricks

- **Estimated Time:** 50 - 60 minutes
- **Lab files:** The files for this lab are located in the *Instructions\Labs\06* folder.

Lab overview

This lab teaches you how to use various Apache Spark DataFrame methods to explore and transform data in Azure Databricks. The student will learn how to perform standard DataFrame methods to explore and transform data. They will also learn how to perform more advanced tasks, such as removing duplicate data, manipulate date/time values, rename columns, and aggregate data.

Lab objectives

After completing this lab, you will be able to:

- Use DataFrames in Azure Databricks to explore and filter data
- Cache a DataFrame for faster subsequent queries
- Remove duplicate data
- Manipulate date/time values
- Remove and rename DataFrame columns
- Aggregate data stored in a DataFrame

Module summary

module-summary

In this module, students have learned how to harness the power of Apache Spark and powerful clusters running on the Azure Databricks platform in order to run large data engineering workloads in the cloud combined with Azure Synapse Analytics.

Learning objectives

In this module, you have learned to:

- Describe Azure Databricks
- Read and write data in Azure Databricks
- Work with DataFrames in Azure Databricks
- Work with DataFrames advanced methods in Azure Databricks

Post Course Review

After the course, consider visiting the website that explores the **Azure Databricks concepts²⁹** and architectures where the associated documentation goes more in depth about architectures and concepts related to Azure Databricks.

Important

Remember

Should you be working in your own subscription while running through this content, it is best practice at the end of a project to identify whether or not you still need the resources you created. Resources left running can cost you money.

You can delete resources one by one, or just delete the resource group to get rid of the entire set.

²⁹ <https://docs.microsoft.com/en-us/azure/databricks/getting-started/concepts>

Answers

Question 1

How many drivers does a Cluster have?

- Only one.
- Two, running in parallel.
- Configurable between one and eight.

Explanation

Correct. A Cluster has one and only one driver.

Question 2

Spark is a distributed computing environment. Therefore, work is parallelized across executors. At which two levels does this parallelization occur?

- The Executor and the Slot.
- The Driver and the Executor.
- The Slot and the Task.

Explanation

Correct. The first level of parallelization is the Executor - a Java virtual machine running on a node, typically, one instance per node. Each Executor has a number of Slots to which parallelized Tasks can be assigned to it by the Driver.

Question 3

What type of process are the driver and the executors?

- Java processes.
- Python processes.
- C++ processes.

Explanation

Correct. The driver and the executors are Java processes.

Question 1

How do you list files in DBFS within a notebook?

- ls /my-file-path.
- %fs dir /my-file-path.
- %fs ls /my-file-path.

Explanation

Correct. You added the file system magic to the cell before executing the ls command.

Question 2

How do you infer the data types and column names when you read a JSON file?

- `spark.read.option("inferSchema", "true").json(jsonFile)`.
- `spark.read.inferSchema("true").json(jsonFile)`.
- `spark.read.option("inferData", "true").json(jsonFile)`.

Explanation

Correct. This approach is the correct way to infer the file's schema.

Question 1

Which DataFrame method do you use to create a temporary view?

- `createTempView()`
- `createTempViewDF()`
- `createOrReplaceTempView()`

Explanation

Correct. You use this method to create temporary views in DataFrames.

Question 2

How do you create a DataFrame object?

- Introduce a variable name and equate it to something like `myDataFrameDF =`
- Use the `createDataFrame()` function
- Use the `DF.create()` syntax

Explanation

Correct. This approach is the correct way to create DataFrame objects.

Question 3

How do you cache data into the memory of the local executor for instant access?

- `.save().inMemory()`
- `.inMemory().save()`
- `.cache()`

Explanation

Correct. The `cache()` method is an alias for `persist()`. Calling this moves data into the memory of the local executor.

Question 4

What is the Python syntax for defining a DataFrame in Spark from an existing Parquet file in DBFS?

- `IPGeocodeDF = parquet.read("dbfs:/mnt/training/ip-geocode.parquet")`
- `IPGeocodeDF = spark.read.parquet("dbfs:/mnt/training/ip-geocode.parquet")`
- `IPGeocodeDF = spark.parquet.read("dbfs:/mnt/training/ip-geocode.parquet")`

Explanation

Correct. This syntax is correct.

Question 1

Which method for renaming a DataFrame's column is incorrect?

- `df.select(col("timestamp").alias("dateCaptured"))`
- `df.alias("timestamp", "dateCaptured")`
- `df.toDF("dateCaptured")`

Explanation

Correct. The DataFrame does not contain an alias method for a column.

Question 2

You need to find the average of sales transactions by storefront. Which of the following aggregates would you use?

- `df.select(col("storefront")).avg("completedTransactions")`
- `df.groupBy(col("storefront")).avg(col("completedTransactions"))`
- `df.groupBy(col("storefront")).avg("completedTransactions")`

Explanation

Correct. The syntax shown groups the data by the storefront Column, then calculates the average value of completed sales transactions.

Module 4 Explore, transform, and load data into the Data Warehouse using Apache Spark

Module introduction

module-introduction

In this module, you will learn how to perform data engineering with Azure Synapse Apache Spark pools, which enable you to boost the performance of big-data analytic applications by in-memory cluster computing. You will ingest data with Apache Spark notebooks, transform data with DataFrames, integrate SQL and Apache Spark pools in Azure Synapse Analytics, while monitoring and managing the workloads.

Learning objectives

In this module, you will:

- Understand big data engineering with Apache Spark pools in Azure Synapse Analytics
- Ingest data with Apache Spark notebooks in Azure Synapse Analytics
- Transform data with DataFrames in Apache Spark pools in Azure Synapse Analytics
- Integrate SQL and Apache Spark pools in Azure Synapse Analytics
- Monitor and manage data engineering workloads with Apache Spark pools in Azure Synapse Analytics

Understand big data engineering with Apache Spark in Azure Synapse Analytics

Introduction

Understand the use cases of data engineering with Apache Spark in Azure Synapse Analytics.

After completing this module, you will be able to:

- Describe what Apache Spark in Azure Synapse Analytics is
- Describe how Apache Spark in Azure Synapse Analytics works
- Understand when to use Apache Spark in Azure Synapse Analytics.

Prerequisite

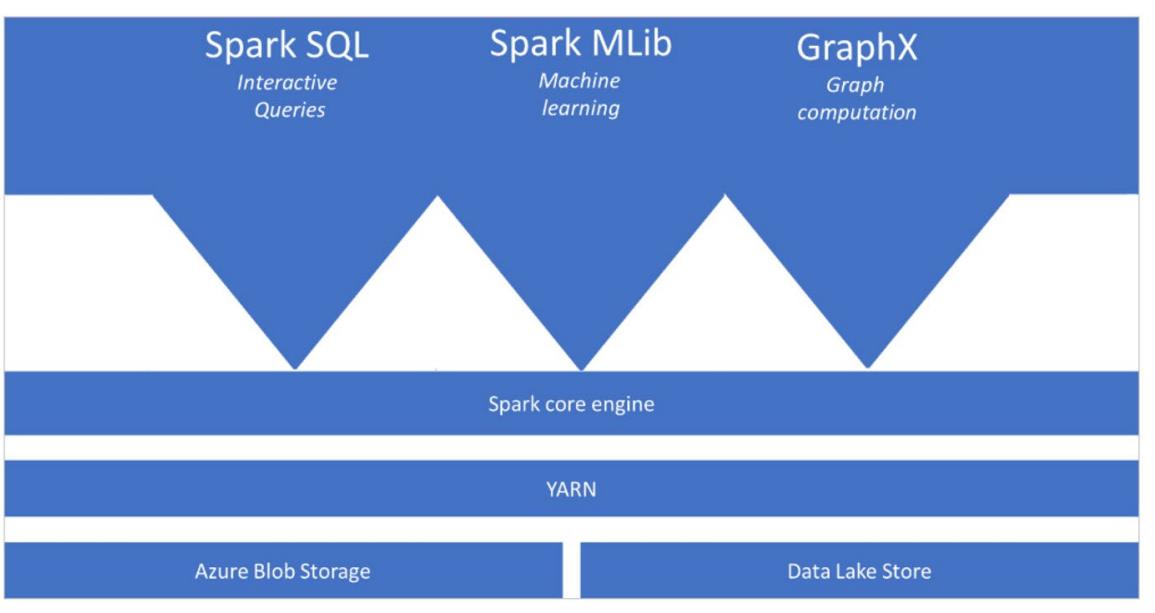
Before taking this module, it is recommended that you complete the following modules:

- Azure Data Fundamentals
- Introduction to Azure Data Factory
- Introduction to Azure Synapse Analytics

What is an Apache Spark pool in Azure Synapse Analytics

Apache Spark is a parallel processing framework that supports in-memory processing to boost the performance of big-data analytics applications. The Apache Spark core engine is a distributed execution engine, of which its resources are managed by the YARN (Yet Another Resource Negotiator) layer to ensure proper use of the distributed engine to process the Spark queries and jobs. You can have different functionalities sit on top of the Apache Spark core engine that you add as your analytics requires.

Examples include adding Apache Spark SQL to perform interactive queries for exploratory data analysis, Spark MLlib for machine learning, and GraphX for graph computations. The functionalities enable you to set up diverse workloads on a single platform.



Apache Spark in Azure Synapse Analytics is one of Microsoft's implementations of Apache Spark in the cloud. Azure Synapse Analytics makes it easy to create and configure a serverless Apache Spark pool in Azure. Spark pools in Azure Synapse Analytics are compatible with Azure Storage and Azure Data Lake Generation 2 Storage. Therefore, you can use Spark pools to process your data stored in Azure.

Its key strength is that it resides on the same platform that contains features for building data warehouses with SQL pools, creating data integration solutions with Azure Synapse Pipelines, and integrating with services such as Microsoft Power BI. This integration makes it a compelling choice to use Azure Synapse Analytics as it can act as your one stop shop for all of your analytical solutions.

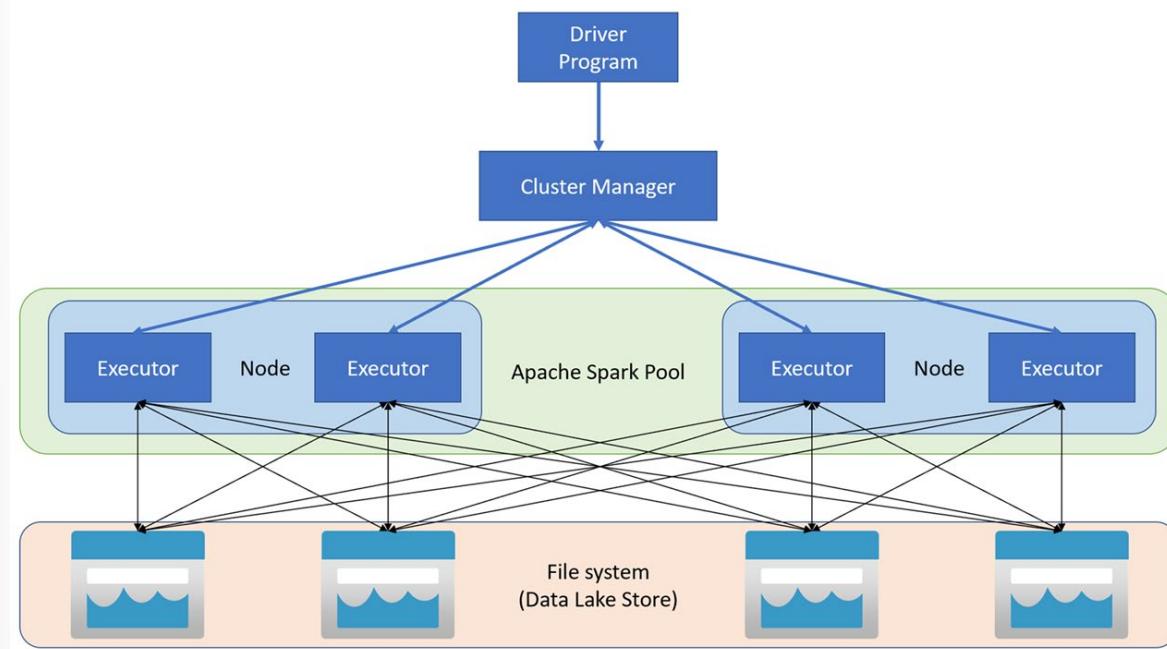
The benefits of using Apache Spark pools in Azure Synapse Analytics are as follows:

- **Speed and Efficiency:** There is a quick start-up time for nodes and automatic shut-down when instances are not used within 5 minutes after the last job, unless there is a live notebook connection.
- **Ease of creation:** Creating an Apache Spark pool can be done through the Azure portal, PowerShell, or .NET SDK for Azure Synapse Analytics.
- **Ease of use:** Within the Azure Synapse Analytics workspace, you can connect directly to the Apache Spark pool and interact with the integrated notebook experience, or use custom notebooks derived from RStudio. Notebook integration helps you develop interactive data processing and visualization pipelines.
- **REST APIs:** In order to monitor and submit jobs remotely, you can use Apache Livy as a Rest API for the Spark job server. Apache Livy is a service that enables interaction with an Spark cluster over a REST interface to enable submission of Spark jobs, snippets of Spark code, synchronous or asynchronous result retrieval, and Spark Context management.
- **Integration with third-party Integrated Development Environments (IDEs):** Azure Synapse Analytics provides an IDE for IntelliJ to create and submit applications to the Apache Spark pool
- **Pre-loaded Anaconda libraries:** Over 200 Anaconda libraries are pre-installed on the Spark pool in Azure Synapse Analytics.
- **Scalability:** Possibility for autoscale, so that pools can be scaled up/down as required, by adding or removing nodes.

How do Apache Spark pools work in Azure Synapse Analytics

Within Azure Synapse Analytics, Apache Spark applications run as independent sets of processes on a pool, coordinated by the `SparkContext` object in your main program (called the driver program). The `SparkContext` can connect to the cluster manager, which allocates resources across applications. The cluster manager is Apache Hadoop YARN. Once connected, Spark acquires executors on nodes in the pool, which are processes that run computations and store data for your application. Next, it sends your application code (defined by JAR or Python files passed to `SparkContext`) to the executors. Finally, `SparkContext` sends tasks to the executors to run.

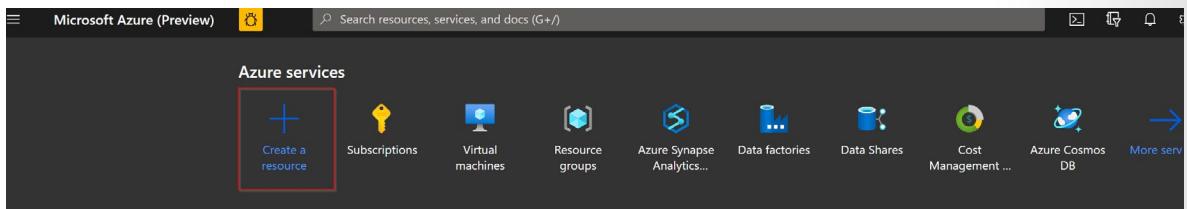
The `SparkContext` runs the user's main function and executes the various parallel operations on the nodes. Then, the `SparkContext` collects the results of the operations. The nodes read and write data from and to the file system. The nodes also cache transformed data in-memory as Resilient Distributed Datasets (RDDs).



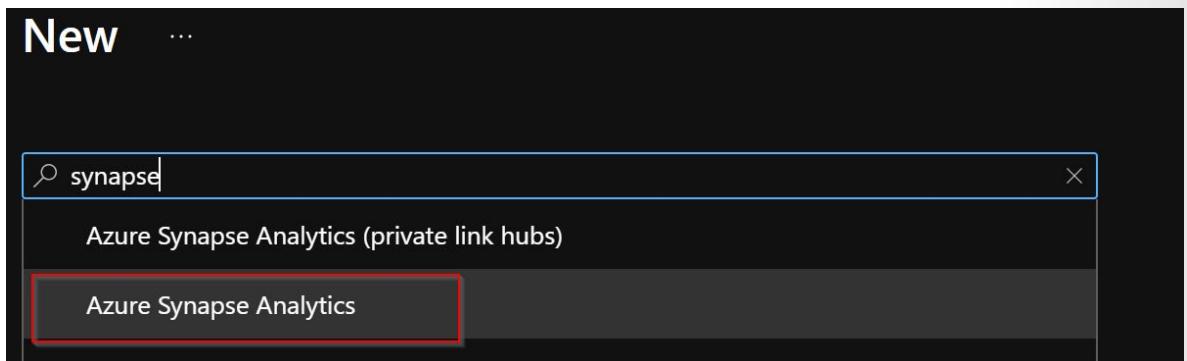
The `SparkContext` connects to the Apache Spark pool and is responsible for converting an application to a directed acyclic graph (DAG). The graph consists of individual tasks that get executed within an executor process on the nodes. Each application gets its own executor processes, which stay up for the duration of the whole application and runs tasks in multiple threads.

The processes are managed automatically when you create an Apache Spark pool in Azure Synapse Analytics. In order to do so, you would have to create an Azure Synapse Analytics workspace first, then you can create a new Apache Spark pool as shown in the following steps.

1. In the Azure portal, select + **Create a resource**



2. In the search environment type **Synapse** and select **Azure Synapse Analytics**:



3. Once you select **Create**, you'll have some parameters to fill out.

Home > New > Azure Synapse Analytics >

Create Synapse workspace

...

* Basics * Security Networking Tags Review + create

Create a Synapse workspace to develop an enterprise analytics solution in just a few clicks.

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 * ⓘ

[REDACTED] ▼

Resource group * ⓘ

[REDACTED] ▼

[Create new](#)

Managed resource group ⓘ

Enter managed resource group name

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 *

Enter workspace name

Region *

West Europe ▼

Select Data Lake Storage Gen2 * ⓘ

From subscription Manually via URL

Account name * ⓘ

[REDACTED] ▼

[Create new](#)

File system name *

[REDACTED] ▼

[Create new](#)

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

[Review + create](#)

< Previous

Next: Security >

- Once you've filled out the parameters, select **Review + create** and wait until the resource gets deployed. Once the Azure Synapse Analytics Workspace resource is created, you are now able to add an Apache Spark pool.
 - In the Azure portal, navigate to the Azure Synapse workspace, and in the overview screen, select **New Apache Spark pool**

The screenshot shows the Azure portal interface for a Synapse workspace named "testworkspacemod". The left sidebar includes links for Overview, Activity log, Access control (IAM), Tags, Settings, SQL Active Directory admin, Properties, and Locks. The main content area displays workspace details such as Resource group, status, location, subscription, and managed virtual network settings. It also lists endpoints for Dedicated SQL, Serverless SQL, and Development. A "New Apache Spark pool" button is visible at the top. The URL in the browser bar is https://testworkspacemod.azuresynapse.net.

6. In the **Create Apache Spark pool** screen, you'll have to specify a couple of parameters including:

- Apache Spark pool name
- Node size
- Autoscale
- Number of nodes

The screenshot shows the 'Create Apache Spark pool' interface on the 'Basics' tab. It includes fields for 'Apache Spark pool name' (with placeholder 'Enter Apache Spark pool name'), 'Node size family' (set to 'MemoryOptimized'), 'Node size' (set to 'Medium (8 vCPU / 64 GB)'), 'Autoscale' (set to 'Enabled'), 'Number of nodes' (set to 3, with a slider ranging from 3 to 10), and an 'Estimated price' section showing '\$40.00 - \$11,000.00'. A warning message at the bottom states: '⚠️ Contact an Owner of the storage account, and verify that the following role assignments have been made:' followed by two bullet points about assigning MSI roles to Storage Blob Data Contributor. Below this is a note: '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'. Navigation buttons at the bottom include 'Review + create', '< Previous', and 'Next: Additional settings >'.

7. Once you've filled out the basic parameters, you could navigate to the **additional settings** tab to customize extra configuration parameters including autoscale and component versions.

Create Apache Spark pool +

Basics Additional settings Tags Summary

Customize additional configuration parameters including autoscale and component versions.

Auto-pause

Enter required settings for this Apache Spark pool, including setting auto-pause and picking versions.

Auto-pause * Enabled Disabled

Number of minutes idle *

Component versions

Select the Apache Spark version for your Apache Spark pool.

Apache Spark *	<input type="text" value="2.4"/>
Python	3.6.1
Scala	2.11.12
Java	1.8.0_222
.NET Core	3.1
.NET for Apache Spark	0.10.0
Delta Lake	0.6.1

Packages

Upload environment configuration file. [Learn more](#)

File upload Browse

Review + create < Previous Next: Tags >

- Once you've finished setting up the parameters, you can select **Review + create** and the Spark pool will be created.

When do you use Apache Spark pools in Azure Synapse Analytics

With the variety of Apache Spark data services that are available on Azure, the following table outlines where Azure Synapse Analytics Apache Spark pools fit in the ecosystem.

	Apache Spark	Azure HDInsight	Azure Databricks	Apache Spark for Azure Synapse
What	Is an open-source, memory-optimized system for managing big data workloads	Microsoft implementation of open-source Apache Spark managed within the realms of Azure	An advanced analytics managed Apache Spark-as-a-Service solution	Embedded Apache Spark capability within Azure Synapse Analytics residing on the same platform that contains data warehouses and data integration capabilities, as well as integrating with other Azure services
When	When you want to benefits of Apache Spark for big data processing and/or data science work without the Service Level Agreements (SLA's) of a provider	When you want to benefits of OSS Spark with the SLA of a provider	Provides an end-to-end data engineering and data science solution and management platform	Enables organizations without existing Apache Spark implementations to fire up an Apache Spark cluster to meet data engineering needs without the overhead of the other Apache Spark platforms listed
Who	Open-source Professionals	Open-source Professionals wanting SLA's and Microsoft Data Platform experts	Data Engineers and Data Scientists working on big data projects every day	Data Engineers, Data Scientists, Data Platform experts and Data Analysts
Why	To overcome the limitations of symmetric multi-processing (SMP) systems imposed on big data workloads	To take advantage of the OSS Big Data Analytics platform with SLA's in place to ensure business continuity	It provides the ability to create and manage an end-to-end big data/data science project using one platform	It provides the ability to scale efficiently with Apache Spark clusters within a one stop shop analytical platform to meet your needs.

The different data services on Azure that have implementations of Apache Spark as mentioned in the table are explained below:

Apache Spark pools in Azure Synapse Analytics:

Apache Spark pools in Azure Synapse Analytics has Apache Spark capabilities embedded. For organizations that don't have existing Apache Spark implementations yet, Apache Spark pools in Azure Synapse Analytics provide the functionality to spin up an Apache Spark cluster to meet data engineering needs

without the overhead of the other Apache Spark platforms. Data engineers, data scientist, data platform experts, and data analyst can come together within Azure Synapse Analytics where the Apache Spark cluster is running to quickly collaborate on various analytical solutions.

Apache Spark for Azure Synapse:

Apache Spark is an open-source, memory-optimized system for managing big data workloads, which is used when you want a Spark engine for big data processing or data science, and you don't mind that there is no service level agreement provided to keep the services running. Usually, it is of interest of open-source professionals and the reason for Apache Spark is to overcome the limitations of what was known as SMP systems for big data workloads.

Azure HDInsight (HDI):

HDI is an implementation by Microsoft of open-source Apache Spark, managed on the Azure Platform. You can use HDI for an Apache Spark environment when you are aware of the benefits of Apache Spark in its OSS form, but you want a Service Level Agreement (SLA). This implementation is usually of interest to open-source professionals needing an SLA and data platform experts experienced with Microsoft products and services.

Azure Databricks:

Azure Databricks is a managed Apache Spark-as-a-Service propriety solution that provides an end-to-end data engineering/data science platform. Azure Databricks is of interest for many data engineers and data scientists working on big data projects today. It provides the platform in which you can create and manage the big data/data science projects all on one platform.

These services are not mutually exclusive. It is common to find customers who use a combination of these technologies working together.

Knowledge check

Question 1

What is an element of an Apache Spark Pool in Azure Synapse Analytics?

- Azure HDInsight.
- Apache Spark Console.
- Spark Instance.

Question 2

In order to create an Apache Spark pool in Azure Synapse Analytics, what needs to be created first?

- Azure Synapse Analytics Workspace.
- Azure Synapse Studio.
- An Apache Spark Instance.

Summary

In this module, you have learned about the use cases of data engineering with Apache Spark in Azure Synapse Analytics.

Now that you have completed this module, you will be able to:

- Describe what Apache Spark in Azure Synapse Analytics is
- Describe how Apache Spark in Azure Synapse Analytics works
- Understand when to use Apache Spark in Azure Synapse Analytics.

Ingest data with Apache Spark notebooks in Azure Synapse Analytics

Introduction

In this module, you will learn how to ingest data using Apache Spark notebooks in Azure Synapse Analytics.

After completing this module, you will be able to:

- Understand Apache Spark notebooks
- Understand the use-cases for Apache Spark notebooks
- Create an Apache Spark notebook in Azure Synapse Analytics
- Understand the supported languages in Apache Spark notebooks
- Develop Apache Spark notebooks
- Run Apache Spark notebooks
- Load data into Apache Spark notebooks
- Save Apache Spark notebooks

Prerequisites

Before taking this module, it is recommended that you complete the following Microsoft Learn content:

- Azure Data Fundamentals
- Introduction to Azure Data Factory
- Introduction to Azure Synapse Analytics

Introduction to spark notebooks

Azure Synapse Studio comes with an integrated notebook experience. The Apache Spark notebooks in Azure Synapse Studio are a web interface that enables you to create, edit, and transform data within them.

It provides a live coding experience, that also enables you to include visualizations and narrative text within the notebook. Notebooks are a great way to explore and validate some of the ideas you might have about the data you are working with. Notebooks enable you to experiment with your data so you can gain some useful insights about the data.

The look and feel of the integrated notebook experience within Azure Synapse Studio is similar to that of Jupyter notebooks in Azure Machine Learning Service.

If you navigate to the Azure Synapse Studio environment, you can find the notebooks in the Development hub of the studio experience.

To access the studio environment, you can navigate to the Azure Synapse Analytics Workspace and launch the studio.

You'll also find that there are some notebook examples available through the Knowledge Center.

Notebooks also contain features that can aid your development as you write code. Some of these features include:

- Support for highlighting language syntax

- Syntax error support
- Syntax code completion
- The ability to export results

To ingest data through notebooks, you can use a linked service already defined in the Azure Synapse workspace. An example could be a linked service defined for an Azure Storage account. By using a linked service, the keys and configurations associated with the storage account are stored in the linked service, and this information is used when you use the notebook to ingest data into an Apache Spark DataFrame from the storage account on which the linked service is defined.

There is an alternative way you can work with data. Every Azure Synapse workspace has an associated primary storage account defined when it is created. If you intend to browse data stored in the primary storage account, there's no need to provide the secrets or keys. In the Data tab on the left hand-side in the Synapse workspace, you can right-click on a file stored in the primary storage account, and then select **New notebook** to open a notebook with the data from the file generated.

Notebooks also enable you to write multiple languages in one notebook by using the magic commands expressed by using the %<Name of Language> syntax. As a result you could create a temporary table to store ingested data within the notebook, and then use the magic command to enable multiple languages to work with this data.

With an Azure Synapse Studio notebook, you can:

- Get started with zero setup effort.
- Keep data secure with built-in enterprise security features.
- Analyze data across raw formats (CSV, txt, JSON, etc.), processed file formats (parquet, Delta Lake, ORC, etc.), and SQL tabular data files against Apache Spark and SQL.
- Be productive with enhanced authoring capabilities and built-in data visualization.

Understand the use-cases for spark notebooks

There are various use cases that make using notebooks compelling within Azure Synapse Analytics

To perform exploratory data analysis using a familiar paradigm

Many data engineers and data scientist work with Apache Spark in various incarnations, be it Microsoft Azure HDInsight, Azure Databricks, or even open-source Apache Spark. The notebooks that are available within Azure Synapse Analytics contain many of the features that these professionals are used to when working with notebooks to explore the data within their organizations. It enables data engineers and scientists to quickly launch a notebook to explore data without the need to learn a new tool, while taking advantage of the inherent integration that the notebooks in Azure Synapse Analytics has with other aspects of the product.

To integrate notebooks as part of a broader data transformation process

Running complex or large transformation on an Apache Spark cluster is at times far more efficient than trying to perform the transformation using traditional relational Transact-SQL methods. You can use notebooks to perform transformations using Apache Spark, and then integrate this transformation into a

broader extract, transform, and load (ETL) process by integrating the notebook into an ETL tool such as Azure Data Factory, or Azure Synapse pipelines.

You wish to perform advance analytics using notebooks with Azure Machine Learning Services

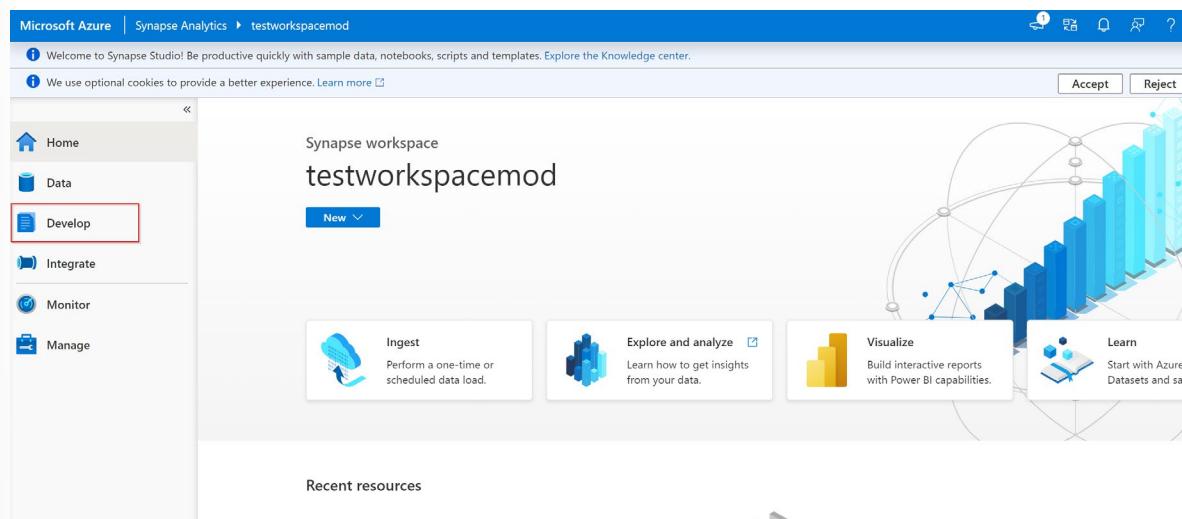
You can create Apache Spark tables in a notebook to connect to a linked service for Azure Machine Learning Services to perform various advanced analytical tasks from classical machine learning to deep learning, both supervised and unsupervised. Whether you prefer to write Python or R code with the SDK or work with no-code/low-code options in Azure Machine Learning Studio, you can build and train machine learning and deep-learning models using the notebooks, and track their activity in an Azure Machine Learning Workspace.

Exercise: Create a spark notebook in Azure Synapse Analytics

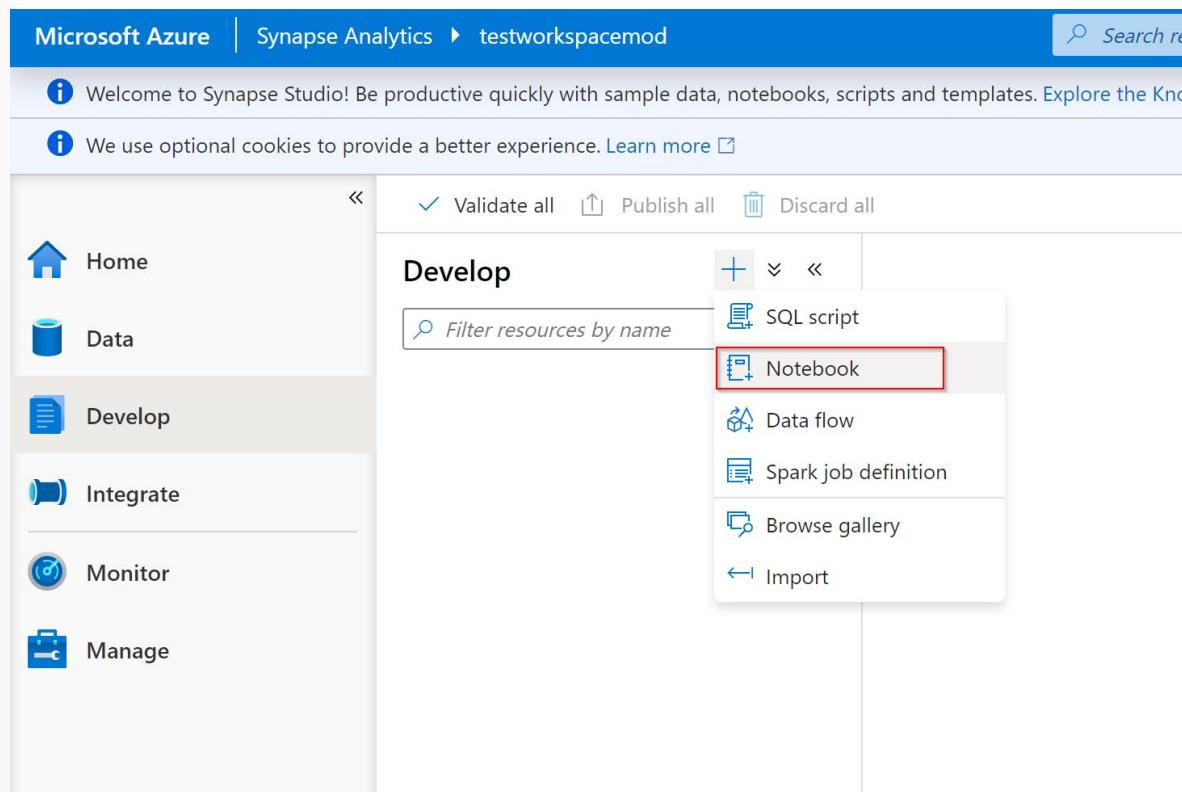
In the Azure portal, navigate to the Azure Synapse workspace you want to use, and select **Open Synapse Studio**.

The screenshot shows the Azure Synapse workspace interface for 'testworkspacemod'. The left sidebar contains navigation links for Overview, Activity log, Access control (IAM), Tags, Settings (SQL Active Directory admin, Properties, Locks), Analytics pools (SQL pools, Apache Spark pools), Security (Firewalls, Managed identities), Monitoring (Alerts, Metrics), Automation (Tasks (preview)), Support + troubleshooting, and New support request. The main content area displays workspace details: Resource group (testmod), Status (Succeeded), Location (West US 2), Subscription (MSFT CSA Internal Subscription), Subscription ID (0adaef70-068b-40c6-a50e-71ea7a1577b4), Managed virtual network (No), Managed Identity object ID (0adaef70-068b-40c6-a50e-71ea7a1577b4), Workspace web URL (<https://web.azuresynapse.net?workspace=%2fsubscriptions%2f0adaef70-068b-40c6-a50e-71ea7a1577b4>), and Tags (Click here to add tags). Below these details is a 'Getting started' section with a 'Open Synapse Studio' button (highlighted by a red box) and a 'Read documentation' link. The 'Analytics pools' section lists 'SQL pools' (Built-in, Serverless) and 'Apache Spark pools' (sparkpoolmod, Apache Spark pool, highlighted by a red box). A search bar at the top allows filtering items.

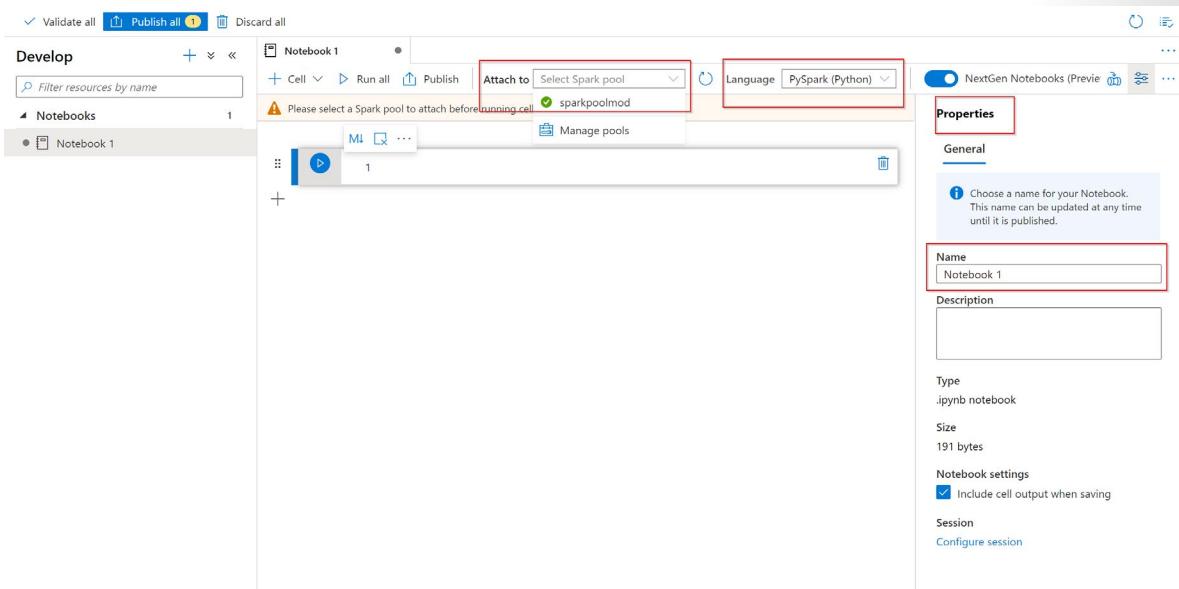
Once the Azure Synapse Studio has launched, select **Develop**.



From there, select the "+" icon, and then select **Notebook**. A new notebook is created and opened with an automatically generated name.



In the **Properties** window, provide a name for the notebook.



On the toolbar, select **Publish**.

If there is only one Apache Spark pool in your workspace, then it's selected by default. Use the drop-down to select the correct Apache Spark pool if none is selected. Add code by using the "+" icon or the "+ Cell" icon and select **Code cell** to input code or **Markdown cell** to input markdown content. When adding a code cell, the default language is PySpark. In the code cell below, you are going to use Pyspark. However, other supported languages are Scala, SQL, and .NET for Spark.

You can select the code below, and paste it into a code cell in your notebook. Run the code, and you'll find that a Spark DataFrame has been created:

```
new_rows = [('CA',22, 45000),("WA",35,65000) ,("WA",50,85000)]
demo_df = spark.createDataFrame(new_rows, ['state', 'age', 'salary'])
demo_df.show()
```

When you want to run a cell, you can use the following methods:

- Press SHIFT + ENTER.
- Select the blue play icon to the left of the cell.
- Select the Run all button on the toolbar.



You can see the Apache Spark pool instance status below the cell you are running and also on the status panel at the bottom of the notebook.

In the output cell, you see the output.

Discover supported languages in spark notebooks

Azure Synapse Analytics Spark pools support various languages. The primary languages available within the notebook environment are:

- PySpark (Python)
- Spark (Scala)
- .NET Spark (C#)
- Spark SQL

It is possible to use multiple languages in one notebook by specifying the language using a magic command at the beginning of a cell.

The following table lists the magic commands to switch cell languages:

Magic command	Language	Description
%%pyspark	Python	Execute a Python query against Spark Context.
%%spark	Scala	Execute a Scala query against Spark Context.
%%sql	SparkSQL	Execute a SparkSQL query against Spark Context.
%%csharp	.NET for Spark C#	Execute a .NET for Spark C# query against Spark Context.

You cannot reference data or variables directly using different languages in an Azure Synapse Studio notebook. If you wish to do this using Spark, you first create a temporary table so that it can be referenced across different languages. Here is an example of how to read a Scala DataFrame in PySpark and SparkSQL using a Spark temp table as a workaround.

- The following code shows you how to read a DataFrame from a SQL pool connector using Scala. It also shows how you can create a temporary table.

```
%%spark
val scalaDataFrame = spark.read.sqlAnalytics("mySQLPoolDatabase.dbo.mySQLPoolTable")
scalaDataFrame.createOrReplaceTempView( "mydataframetable" )
```

- If you want to query the DataFrame in the above example, using Spark SQL, you can add a code cell below the code snippet above, and use the %%sql command. Using the %%sql command enables you to use a SQL statement such as shown below where you would select everything from the 'mydataframetable'.

```
%%sql
SELECT * FROM mydataframetable
```

- If you want to use the data in PySpark, below an example is given using the magic command %%pyspark, in which you create a new Python DataFrame based on the mydataframe table whilst using spark.sql to select everything from that table.

```
%%pyspark
myNewPythonDataFrame = spark.sql("SELECT * FROM mydataframetable")
```

You can use familiar Jupyter magic commands in Azure Synapse Studio notebooks. Review the following list as the current available magic commands:

Available line magics: %lsmagic, %time, %time it

Available cell magics: %%time, %%timeit, %%capture, %%writefile, %%sql, %%pyspark, %%spark, %%csharp

Develop spark notebooks

To develop solutions in a notebook, you work with cells. Cells are individual blocks of code or text that can be run independently or as a group. There are a range of actions that can be performed against a cell including:

- Move a cell.

If you want to align different cells of code and put them in a correct order, the notebook environment gives you the opportunity to move cells to organize your work.

- Delete a cell.

If you have written a cell of code, but no longer need it or it needs to be deleted, the delete functionality can be used within the notebook environment.

- Collapse Cell in and output.

If you want to collapse a cell to check in or output, the functionality is available to you when using the notebook environment in Azure Synapse Studio.

- Undo Cell operations.

When, for example, you need to revoke a cell operation, you can do so within the Azure Synapse Studio notebook environment.

When you use Azure Synapse Studio notebooks, you'll see the integration with the Monaco editor. The Monaco editor enables you to bring an IDE-style IntelliSense to the cell editor. The IDE-style IntelliSense helps you in cases of syntax highlighting, error handling and marking, and automatic code completions to help you write code and identify issues quickly.

The IntelliSense features are at different levels of maturity for different languages. So depending on the language you want to write your code in, there may be different IntelliSense features. The following table shows what is supported based on language.

Languages	Syntax Highlight	Syntax Error Marker	Syntax Code Completion	Variable Code Completion	System Function Code Completion	User Function Code Completion	Smart Indent	Code Folding
PySpark (Python)	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Spark (Scala)	Yes	Yes	Yes	Yes	-	-	-	Yes
SparkSQL	Yes	Yes	-	-	-	-	-	-
.NET for Spark (C#)	Yes	-	-	-	-	-	-	-

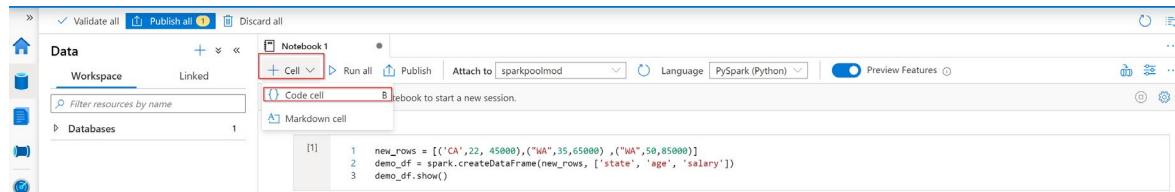
Exercise: Develop spark notebooks

In the following steps, you will explore these notebook development features:

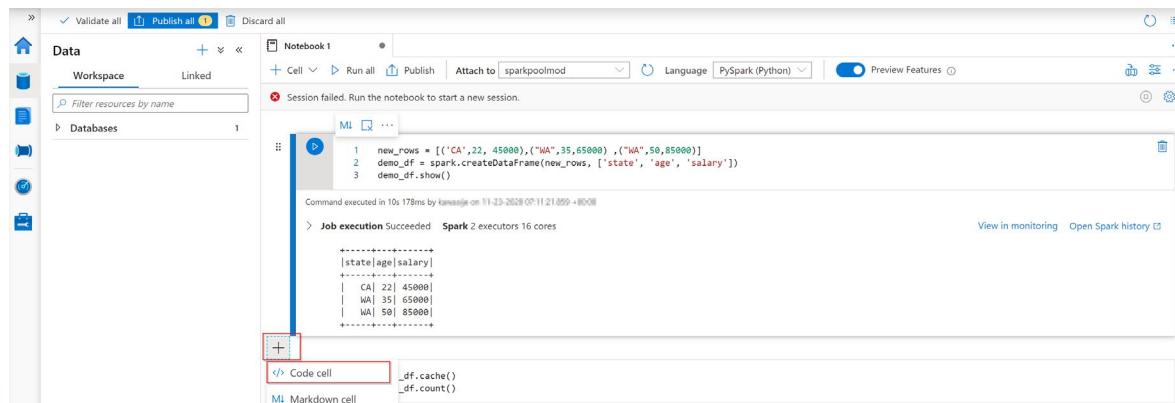
- Adding cells to notebooks
- Undo Cell operations
- Move a cell
- Delete a cell
- Collapse Cell in and output

Adding cells to notebooks

1. Expand the upper left + Cell button, and select **Add code cell** or **Add Markdown Cell**.



2. Hover over the space between two cells and select the + button where you have the option to **Add Code cell** or **Add Markdown Cell**.



3. Use **Shortcut keys under command mode**¹. Press **A** to insert a cell above the current cell. Press **B** to insert a cell below the current cell.

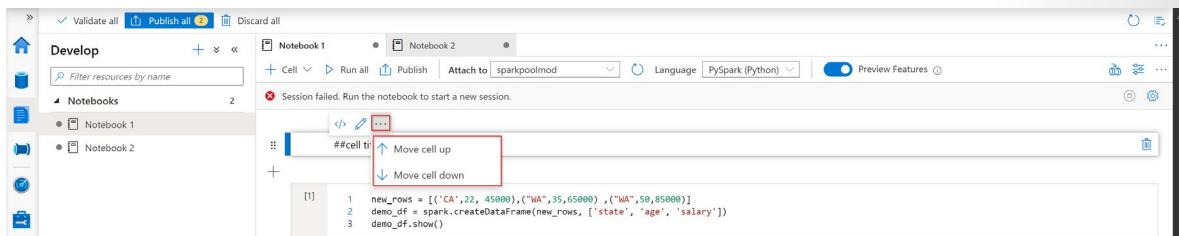
Undo cell operations

1. Press **Ctrl+Z** to revoke the most recent cell operation.

Move a cell

1. Select the ellipses (...) when you hover over the cell you want to move. Select **Move cell up** or **Move cell down** to move the current cell.

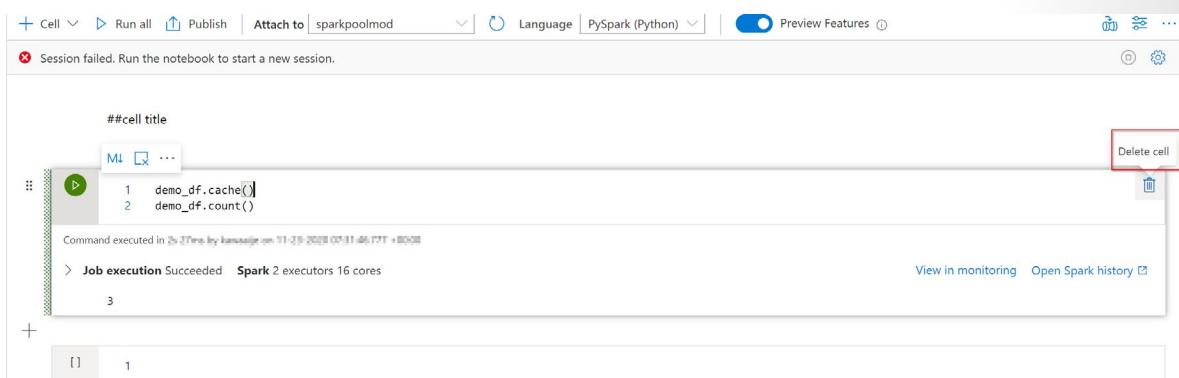
¹ <https://docs.microsoft.com/azure/synapse-analytics/spark/apache-spark-development-using-notebooks#shortcut-keys-under-command-mode>



2. You can also use **shortcut keys under command mode**². Press **Ctrl+Alt+↑** to move up the current cell. Press **Ctrl+Alt+↓** to move the current cell down.

Delete a cell

1. To delete a cell, hover over the far right-side of the cell and a garbage bin icon will appear. Select **Delete cell**.



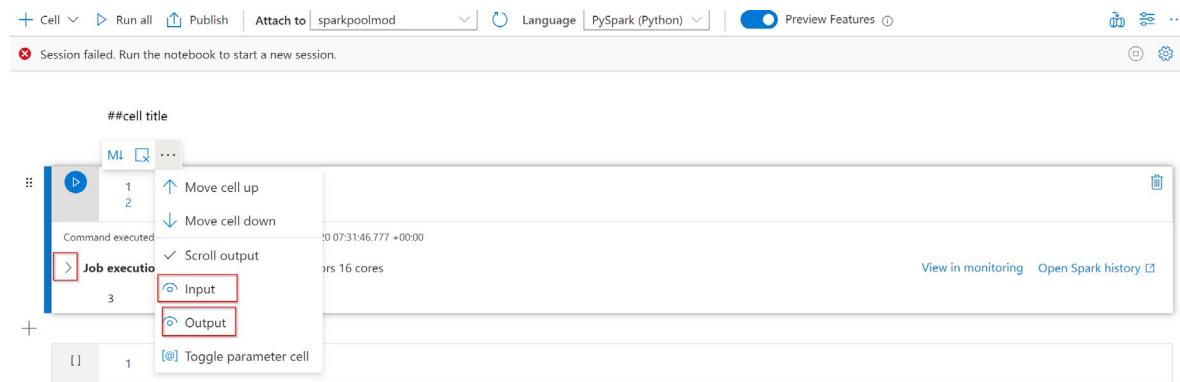
2. You can also use **shortcut keys under command mode**³. Press **D,D** to delete the current cell.

Collapse a cell input and output

1. Hover over the cell you want to see the input or output from.
2. Select the ellipses (...) and you'll find here the input and output option.
3. If you select the eye icon, you can hide the input or output. If you unselect, you'll see the input and output.
4. If you want to see details of the job execution, select the ">" icon.

² <https://docs.microsoft.com/azure/synapse-analytics/spark/apache-spark-development-using-notebooks#shortcut-keys-under-command-mode>

³ <https://docs.microsoft.com/azure/synapse-analytics/spark/apache-spark-development-using-notebooks#shortcut-keys-under-command-mode>



Run spark notebooks

When you want to run notebooks in the Azure Synapse Studio environment, you are able to run the code cells in your notebook individually or all at once. The status and progress of each cell will also be represented in the notebook.

The different functionalities for running a notebook are as follows:

1. Run a Cell

If you want to run once cell of code or text, you can do so through the notebook experience in Azure Synapse Studio.

2. Run all cells

If you have developed code that consists of multiple cells or is combined with text, you can do so through the notebook experience in Azure Synapse Studio.

3. Cancel a running cell

If you hit run, but while the cell is running, you want to cancel one cell run, you can do so within Azure Synapse notebooks.

4. Cell Status indicator

If you want to check the status of a cell while running or completed, you have the possibility to get a status indicator within the notebook experience in Synapse Studio.

5. Spark progress indicator

The Azure Synapse Studio notebook is purely Spark based. Remotely, the code cells that are executed, are executed on the serverless Apache Spark pool. If you want to see the progress of a Spark job, you can see in real time the job execution status below a cell.

The number of tasks per each job or stage help you to identify the parallel level of your spark job. You can also drill deeper to the Spark UI of a specific job (or stage) by selecting the link on the job (or stage) name.

Exercise: Run spark notebooks

In the following steps, you will explore these run features of a notebook:

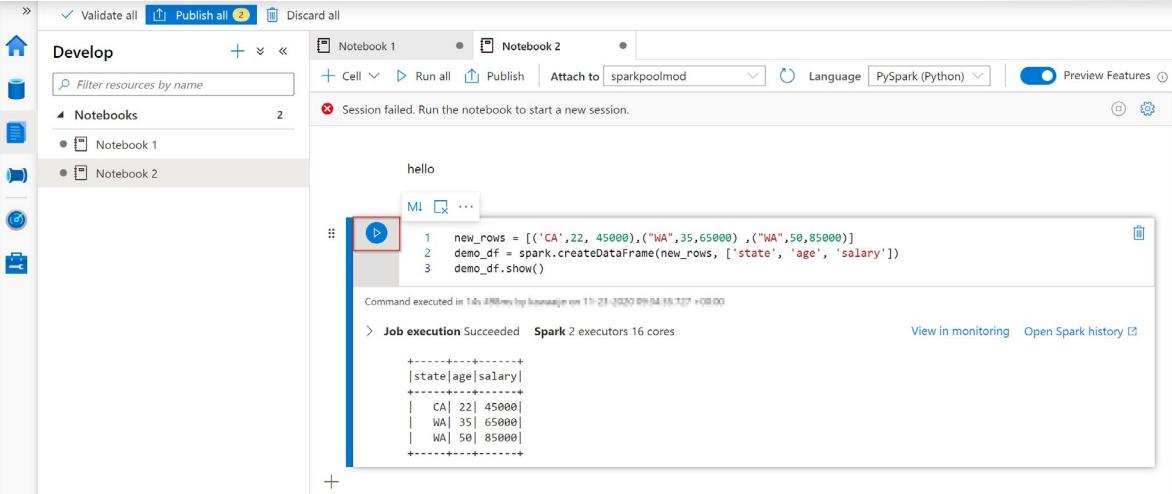
- Run a cell
- Run all cells

- Cancel a running cell
- Cell Status indicator
- Spark progress indicator

1. Run a cell

There are several ways to run the code in a cell.

- Hover on the cell you want to run and select the **Run Cell** button or press **Ctrl+Enter**.



The screenshot shows a Jupyter Notebook interface. On the left is a sidebar with 'Develop' and 'Notebooks' sections. In the main area, 'Notebook 1' is selected. A cell titled 'hello' contains the following Python code:

```

1 new_rows = [('CA',22, 45000),("WA",35,65000 ),("WA",50,85000)]
2 demo_df = spark.createDataFrame(new_rows, ['state', 'age', 'salary'])
3 demo_df.show()

```

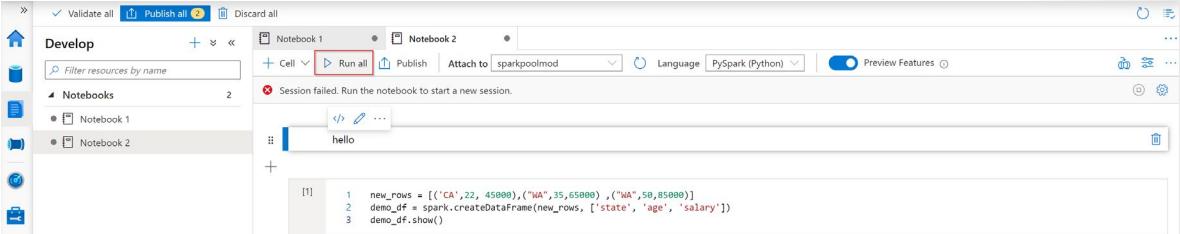
A blue 'Run Cell' button with a play icon is highlighted with a red box. Below the cell, the output shows the command was executed in 14ms and the resulting DataFrame:

state	age	salary
CA	22	45000
WA	35	65000
WA	50	85000

- In addition, you can press **Shift+Enter** to run the current cell and select the cell below. Alternatively press **Alt+Enter** to run the current cell and insert a new cell below.

2. Run all cells

Select the **Run All** button to run all the cells in current notebook in sequence.



The screenshot shows a Jupyter Notebook interface with 'Notebook 1' selected. A cell titled 'hello' contains the same code as before:

```

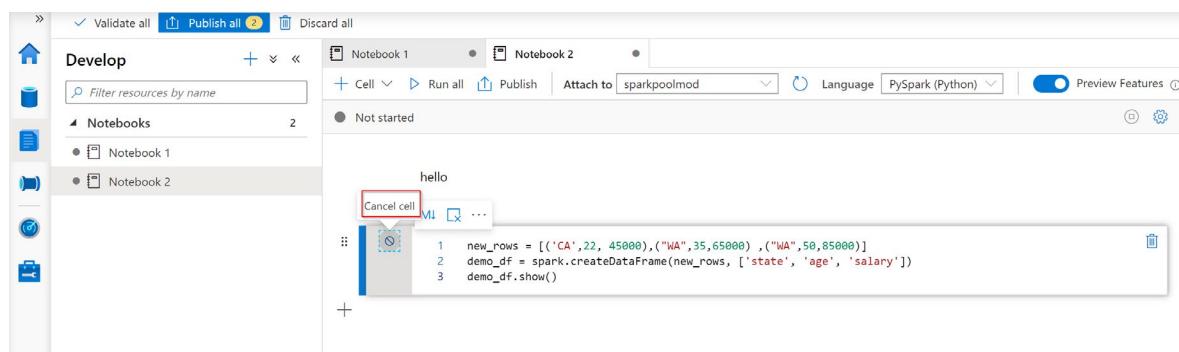
1 new_rows = [('CA',22, 45000),("WA",35,65000 ),("WA",50,85000)]
2 demo_df = spark.createDataFrame(new_rows, ['state', 'age', 'salary'])
3 demo_df.show()

```

The 'Run all' button in the toolbar is highlighted with a red box. The output shows the command was executed in 14ms and the resulting DataFrame.

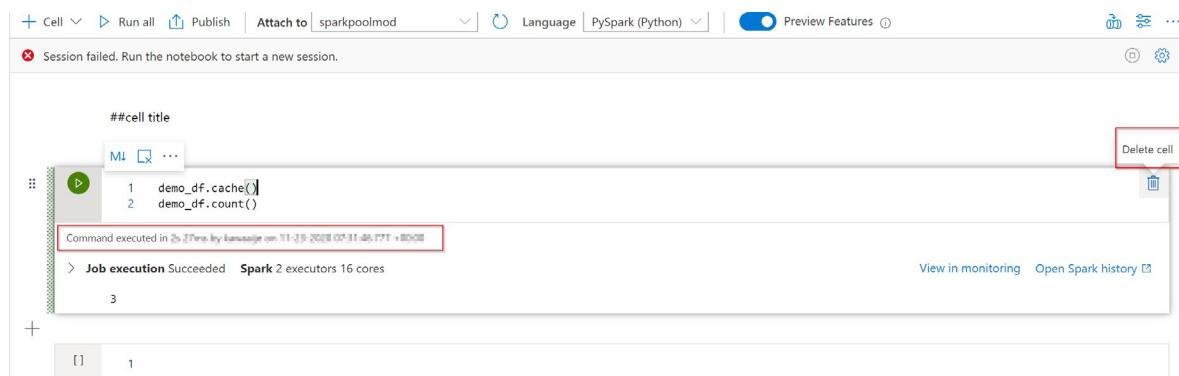
3. Cancel a running cell

In order to cancel a running cell, you need to hover over the cell that is running on the left side while it's running and select "cancel cell".



4. Cell status indicator

A step-by-step cell execution status is displayed beneath the cell to help you see its current progress. Once the cell run is complete, an execution summary with the total duration and end time are shown and kept there for future reference.



5. Spark progress indicator

Azure Synapse Studio notebook is purely Spark based.

Remotely, the code cells that are executed, are executed on the serverless Apache Spark pool. If you want to see the progress of a spark job, you can see in real time the job execution status below a cell. The number of tasks per each job or stage help you to identify the parallel level of your spark job. You can also drill deeper to the Spark UI of a specific job (or stage) via selecting the link on the job (or stage) name.

The screenshot shows the Databricks interface. On the left, the sidebar displays 'Develop' mode, 'Notebooks' section with 'Notebook 1' and 'Notebook 2' selected, and a search bar. The main area shows 'Notebook 2' with a cell containing PySpark code:

```
1 new_rows = [('CA',22, 45000),("WA",35,65000) ,("WA",50,85000)]
2 demo_df = spark.createDataFrame(new_rows, ['state', 'age', 'salary'])
3 demo_df.show()
```

Below the code, it says "Command executed in 18s 41ms log taken at 11-21-2020 10:08:17.912 +08:00". A table titled "Job execution Succeeded" lists three jobs:

ID	Description	Status	Stages	Tasks	Submission Time
> Job 0	showString at NativeMethodAccessorImpl.java:0	Succeeded	1/1	1/1 succeeded	10:08:28 AM, 11/21/2020
> Job 1	showString at NativeMethodAccessorImpl.java:0	Succeeded	1/1	4/4 succeeded	10:08:29 AM, 11/21/2020
> Job 2	showString at NativeMethodAccessorImpl.java:0	Succeeded	1/1	11/11 succeeded	10:08:30 AM, 11/21/2020

At the bottom, the output shows the DataFrame content:

```
+-----+-----+
|state|age|salary|
+-----+-----+
| CA | 22 | 45000 |
| WA | 35 | 65000 |
| WA | 50 | 85000 |
+-----+-----+
```

6. Spark session config

You can specify the timeout duration, the number, and the size of executors to give to the current Spark session in **Configure session**. Restart the Spark session is for configuration changes to take effect. All cached notebook variables are cleared.

Configure session

Session name
Synapse_sparkpoolmod_1606128093

[View in monitoring](#)
[Open Spark UI](#)

Application ID
application_1606128177873_0001

Livy session ID
5

Status
Ready

Attach to *  sparkpoolmod

 sparkpoolmod 
Refresh at 10:51:20 AM

Medium (8 vCores / 56 GB) 3 - 10 nodes
30.00% utilized (1 application)

Available session sizes 

Small	13 executors	Use
Medium	6 executors	Use

Executor size *  Medium (8 vCores, 56GB memory)

Executors *   2

Driver size *  Medium (8 vCores, 56GB memory)

Session timeout (minutes) *  30

[Apply](#) [Cancel](#)

Load data in spark notebooks

In order to ingest data into a notebook, there are several options. Currently it is possible to load data from an Azure Storage Account, and an Azure Synapse Analytics dedicated SQL pool.

Some examples for reading data in a notebook are:

- Read a CSV from Azure Data Lake Store Gen2 as an Apache Spark DataFrame
- Read a CSV from Azure Storage Account as an Apache Spark DataFrame
- Read data from the primary storage account

Example 1: Read a CSV file from an Azure Data Lake Store Gen2 store as an Apache Spark DataFrame.

The following code is used to read a CSV file from an Azure Data Lake Store Gen2 store as an Apache Spark DataFrame.

```
from pyspark.sql import SparkSession
from pyspark.sql.types import *
account_name = "Your account name"
container_name = "Your container name"
relative_path = "Your path"
adls_path = 'abfss://%s@%.dfs.core.windows.net/%s' % (container_name,
account_name, relative_path)

spark.conf.set("fs.azure.account.auth.type.%s.dfs.core.windows.net" %ac-
count_name, "SharedKey")
spark.conf.set("fs.azure.account.key.%s.dfs.core.windows.net" %account_name
,"Your ADLS Gen2 Primary Key")

df1 = spark.read.option('header', 'true') \
.option('delimiter', ',') \
.csv(adls_path + '/Testfile.csv')
```

There are parameter name values that you need to replace in the above code to ensure that it works, including:

- account_name
Replace "Your account name" with the storage account name you wish to use
- container_name
Replace "Your container name" with the storage container you wish to use
- relative_path
Replace "Your path" with the relative path of where the file is stored
- adls_path

The adls_path is defined by passing through the above parameters.

Example 2: Read a CSV file from Azure Storage Account as a Spark DataFrame.

The following code is used to read a CSV file from Azure Storage Account as an Apache Spark DataFrame.

```
from pyspark.sql import SparkSession
from pyspark.sql.types import *

blob_account_name = "Your blob account name"
blob_container_name = "Your blob container name"
blob_relative_path = "Your blob relative path"
blob_sas_token = "Your blob sas token"

wasbs_path = 'wasbs://{}@{}.blob.core.windows.net/{}'.format(blob_container_name, blob_account_name, blob_relative_path)
spark.conf.set('fs.azure.sas.%s.%s.blob.core.windows.net'.format(blob_container_name, blob_account_name), blob_sas_token)

df = spark.read.option("header", "true") \
    .option("delimiter", "|") \
    .schema(schema) \
    .csv(wasbs_path)
```

There are parameter name values that you need to replace in the above code to ensure that it works, including:

- blob_account_name

Replace “Your blob account name” with the name of your blob account.

- blob_container_name

Replace “Your blob container” with the name of the blob container the file is in.

- blob_relative_path

Replace “Your blob relative path” with the name of the relative path pointing to the csv you want to read.

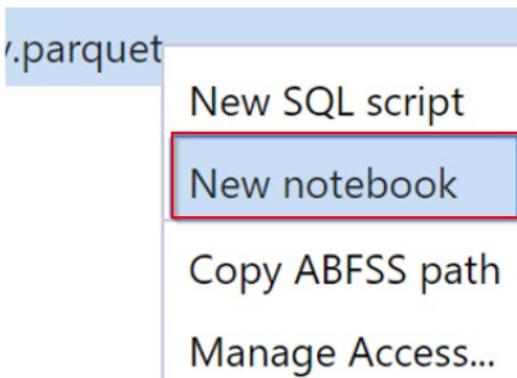
- blob_sas_token

Replace “Your blob sas token” with the blob SAS key.

Example 3: Read data from the primary storage account

The third possibility is to read data from the primary storage account, using the Data tab in the Azure Synapse Studio environment.

If you right-click on a file and select **New notebook**, you will see a new notebook with the data generated.



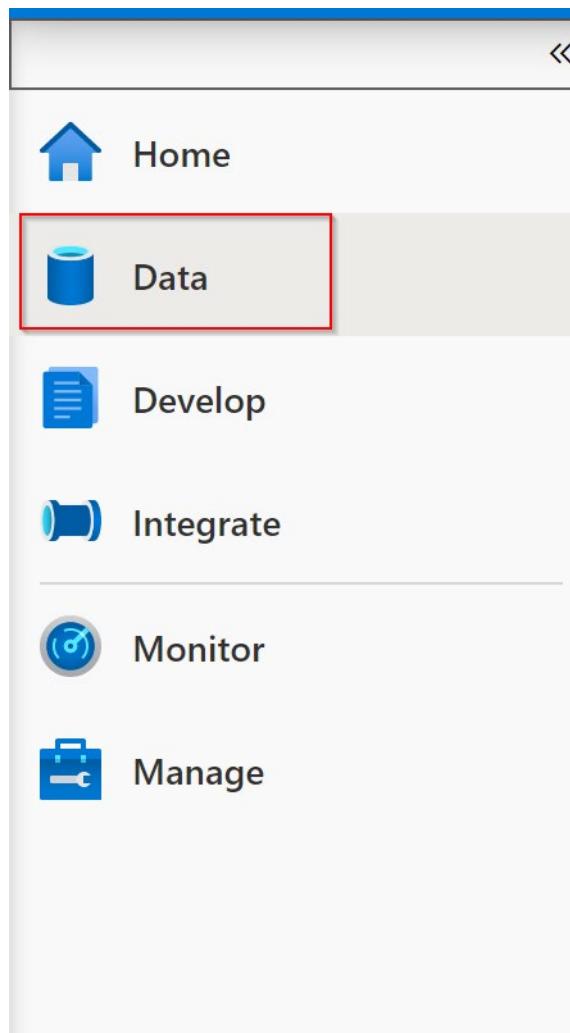
Exercise Load data in spark notebooks

Ingest and explore Parquet files from a data lake with Apache Spark for Azure Synapse

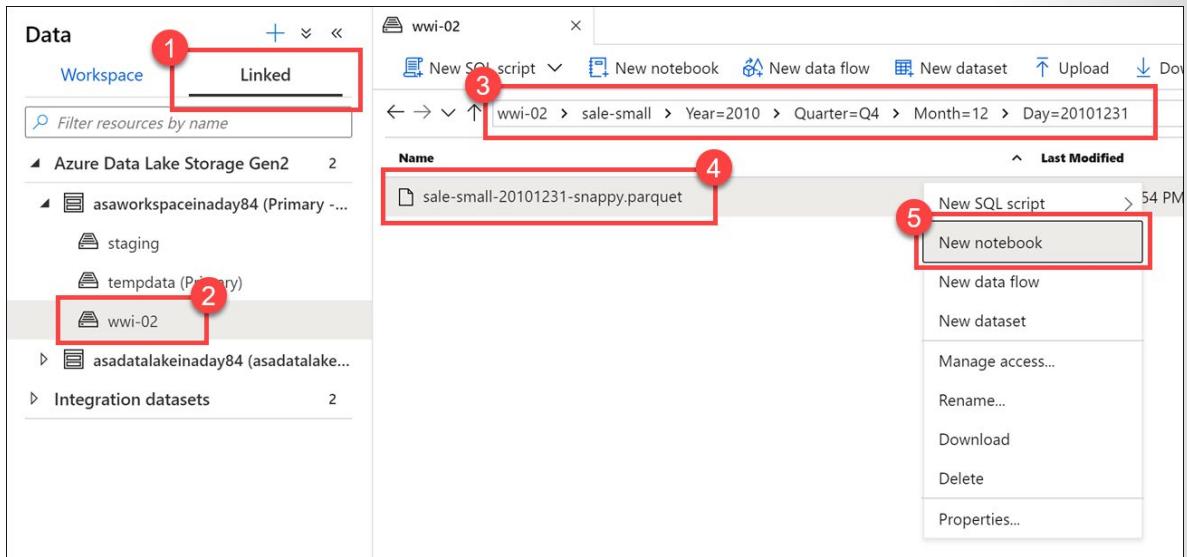
Tailwind Traders has Parquet files stored in their data lake. They want to know how they can quickly access the files and explore them using Apache Spark.

You recommend using the Data hub to view the Parquet files in the connected storage account, then use the *new notebook* context menu to create a new Synapse notebook that loads a Spark DataFrame with the contents of a selected Parquet file.

1. Open Synapse Studio (<https://web.azure-synapse.net/>).
2. Select the **Data** hub.



3. Select the **Linked** tab **(1)** and expand the primary data lake storage account (*the name may differ from what you see here; it is the first storage account listed*). Select the **wwi-02** container **(2)** and browser to the `sale-small/Year=2010/Quarter=Q4/Month=12/Day=20101231` folder **(3)**. Right-click the Parquet file **(4)** and select **New notebook** **(5)**.

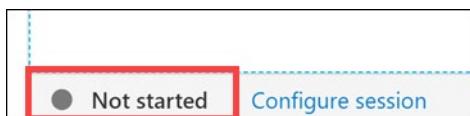


It generates a notebook with PySpark code to load the data in a Spark dataframe and display 100 rows with the header.

4. Make sure the Spark pool is attached to the notebook.



The Spark pool provides the compute for all notebook operations. If we look at the bottom of the notebook, we'll see that the pool has not started. When you run a cell in the notebook while the pool is idle, the pool will start and allocate resources. It is a one-time operation until the pool autopauses from being idle for too long.



The auto-pause settings are configured on the Spark pool configuration in the Manage hub.

We can change the Spark configuration for this session by selecting **Configure session**. Let's do that now.

5. Select **Configure session** at the bottom-left of the notebook.

Configure session

Session name
Synapse_sparkpoolmod_1606128093

[View in monitoring](#)
[Open Spark UI](#)

Application ID
application_1606128177873_0001

Livy session ID
5

Status
Ready

Attach to *  sparkpoolmod

 sparkpoolmod 

Refresh at 10:51:20 AM

Medium (8 vCores / 56 GB) 3 - 10 nodes
30.00% utilized (1 application)

Available session sizes 

Small	13 executors	Use
Medium	6 executors	Use

Executor size *  Medium (8 vCores, 56GB memory)

Executors * 
2

Driver size *  Medium (8 vCores, 56GB memory)

Session timeout (minutes) *  30

[Apply](#) [Cancel](#)

6. Set the number of **Executors** to **3 (1)**, then select **Apply (2)**.

Configure session

Application ID

Livy session ID

Status
Not started

Attach to * ⓘ

SparkPool01

Refresh at 2:04:40 PM

Small (4 vCPU / 28 GB) 3 - 4 nodes
0.00% utilized

Available session sizes ⓘ

Small 3 executors Use

Session timeout * ⓘ

30

1 Executors * ⓘ

3

Executor size * ⓘ

Small (4 vCPU, 28GB memory)

Driver size * ⓘ

Small (4 vCPU, 28GB memory)

2 Apply Cancel

We have just set the number of executors allocated to **SparkPool01** for the session.

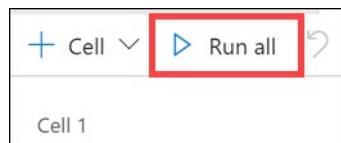
- Add the following code beneath the code in the cell to define a variable named `datalake` whose value is the name of the primary storage account (**replace the REPLACE_WITH_YOUR_DATA LAKE_NAME value with the name of the storage account in line 2**):

```
datalake = 'REPLACE_WITH_YOUR_DATA LAKE_NAME'
```

```
Cell 1
1 %%spark
2 data_path = spark.read.load('abfss://wwi-02@asadatalakeinaday84.dfs.core.windows.net/sale-small/Year=2010/Quarter=Q4/Mo...
3 display(data_path.limit(10))
4
5 datalake = 'asadatalakeinaday84'
```

This variable will be used in a couple cells later on.

- Select **Run all** on the notebook toolbar to execute the notebook.



NOTE: The first time you run a notebook in a Spark pool, Synapse creates a new session. This can take approximately 3-5 minutes.

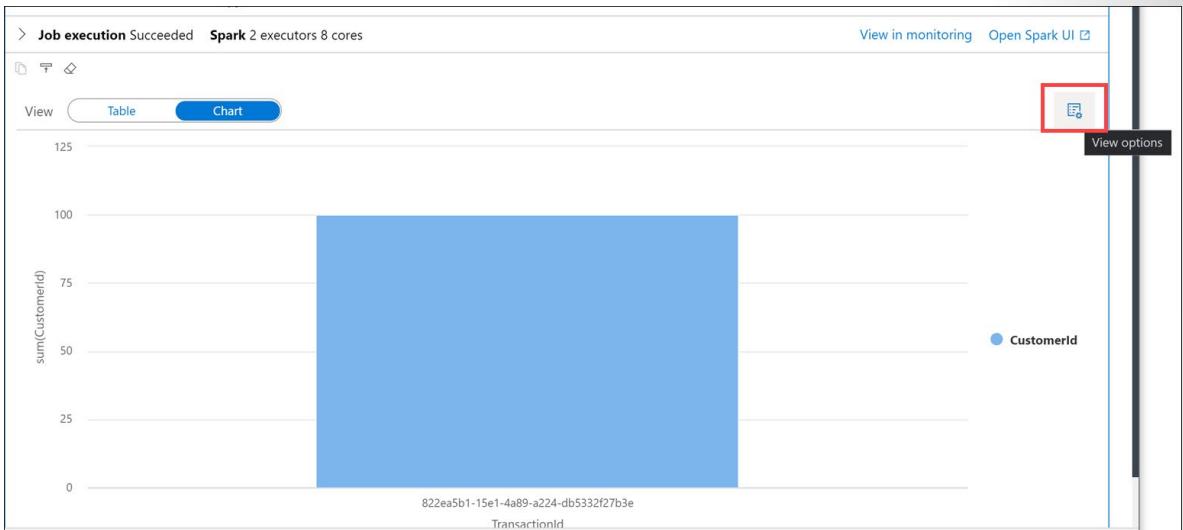
NOTE: To run just the cell, either hover over the cell and select the *Run cell* icon to the left of the cell, or select the cell then type **Ctrl+Enter** on your keyboard.

9. After the cell run is complete, change the View to **Chart** in the cell output.

TransactionId	CustomerId	ProductId	Quantity	Price
822ea5b1-15e1-4a89-a224-db53...	10	64	4	31.28
822ea5b1-15e1-4a89-a224-db53...	10	64	3	31.28
822ea5b1-15e1-4a89-a224-db53...	10	64	2	31.28
822ea5b1-15e1-4a89-a224-db53...	10	3348	2	27.7
822ea5b1-15e1-4a89-a224-db53...	10	1470	1	29.14
822ea5b1-15e1-4a89-a224-db53...	10	1266	1	29.65
822ea5b1-15e1-4a89-a224-db53...	10	48	3	35.55
822ea5b1-15e1-4a89-a224-db53...	10	132	3	34.18
822ea5b1-15e1-4a89-a224-db53...	10	69	3	31.06

By default, the cell outputs to a table view when we use the `display()` function. We see in the output the sales transaction data stored in the Parquet file for December 31, 2010. Let's select the **Chart** visualization to see a different view of the data.

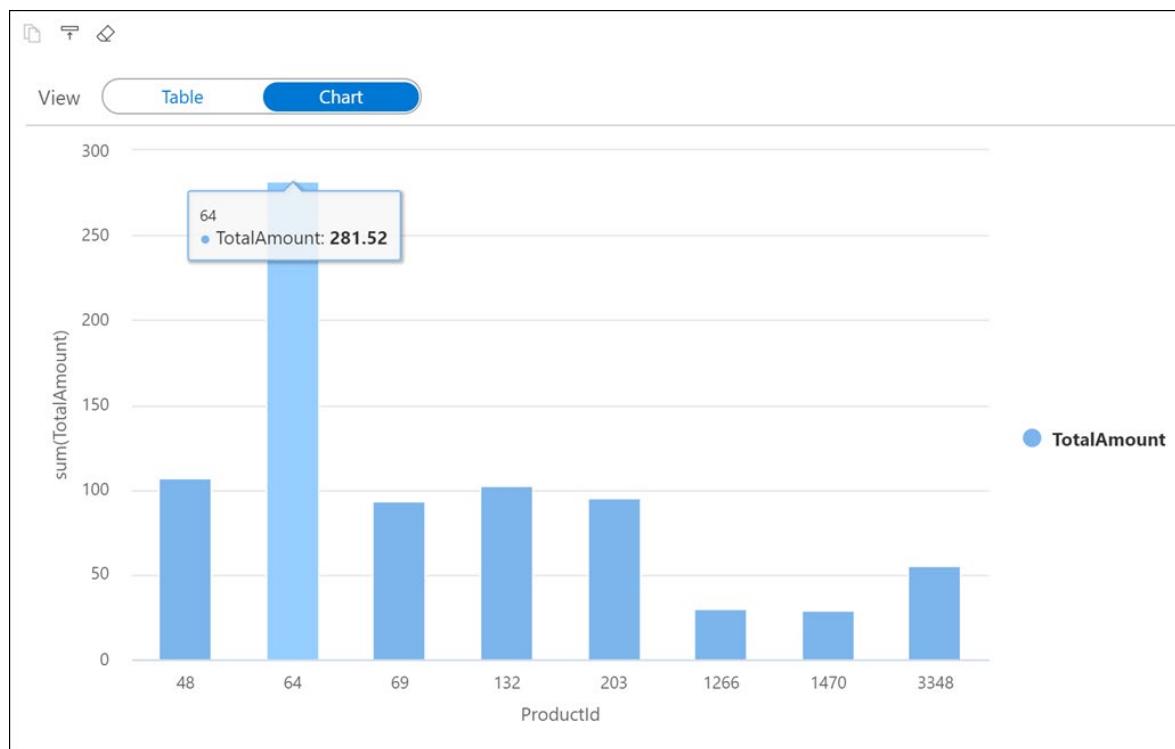
10. Select the **View options** button to the right.



11. Set Key to **ProductId** and Values to **TotalAmount** (1), then select **Apply**.

This screenshot shows a configuration dialog for a chart. It has a "Chart type" dropdown set to "bar chart". Under "Key", "ProductId" is selected. Under "Values", "TotalAmount" is selected. Both of these fields are highlighted with a red box and a red number "1" indicating they are the focus of step 1. Below these, there is a "Series Group" dropdown, an "Aggregation" dropdown set to "SUM", and two checkboxes: "Stacked" and "Aggregation over all results". At the bottom, there are "Apply" and "Cancel" buttons, with the "Apply" button highlighted with a red box and a red number "2" indicating it is the next step to be clicked.

12. The chart visualization is displayed. Hover over the bars to view details.



13. Create a new cell underneath by selecting **{ } Add code** when hovering over the blank space at the bottom of the notebook.



14. The Apache Spark engine can analyze the Parquet files and infer the schema. To do so, enter the below code in the new cell and **run** it:

```
data_path.printSchema()
```

Your output should look as follows:

```
root
 |-- TransactionId: string (nullable = true)
 |-- CustomerId: integer (nullable = true)
 |-- ProductId: short (nullable = true)
 |-- Quantity: short (nullable = true)
 |-- Price: decimal(29,2) (nullable = true)
 |-- TotalAmount: decimal(29,2) (nullable = true)
 |-- TransactionDate: integer (nullable = true)
 |-- ProfitAmount: decimal(29,2) (nullable = true)
```

```
-- Hour: byte (nullable = true)
|-- Minute: byte (nullable = true)
|-- StoreId: short (nullable = true)
```

Apache Spark evaluates the file contents to infer the schema. This automatic inference is sufficient for data exploration and most transformation tasks. However, when you load data to an external resource like a SQL pool table, sometimes you need to declare your own schema and apply that to the dataset. For now, the schema looks good.

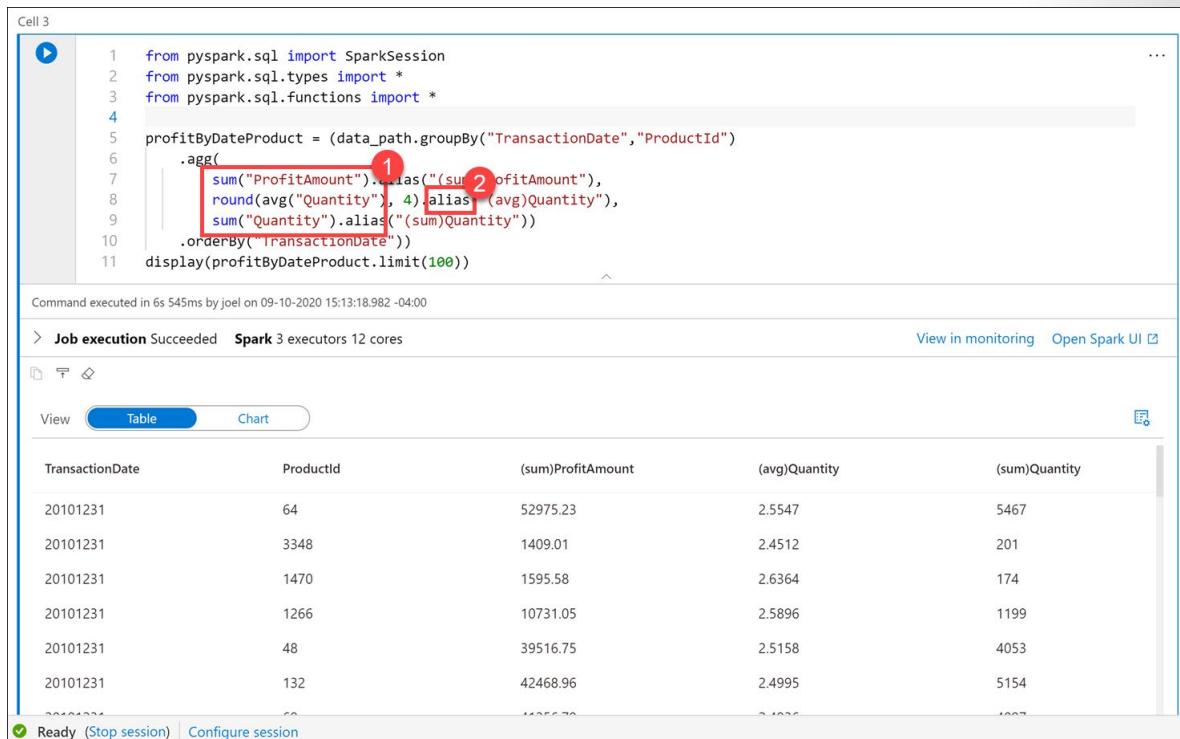
15. Now let's use the dataframe to use aggregates and grouping operations to better understand the data. Create a new cell and enter the following, then **run** the cell:

```
from pyspark.sql import SparkSession
from pyspark.sql.types import *
from pyspark.sql.functions import *

profitByDateProduct = (data_path.groupBy("TransactionDate", "ProductId")
    .agg(
        sum("ProfitAmount").alias("(sum)ProfitAmount"),
        round(avg("Quantity"), 4).alias("(avg)Quantity"),
        sum("Quantity").alias("(sum)Quantity"))
    .orderBy("TransactionDate"))
display(profitByDateProduct.limit(100))
```

We import required Python libraries to use aggregation functions and types defined in the schema to successfully execute the query.

The output shows the same data we saw in the chart above, but now with `sum` and `avg` aggregates (1). Notice that we use the `alias` method (2) to change the column names.



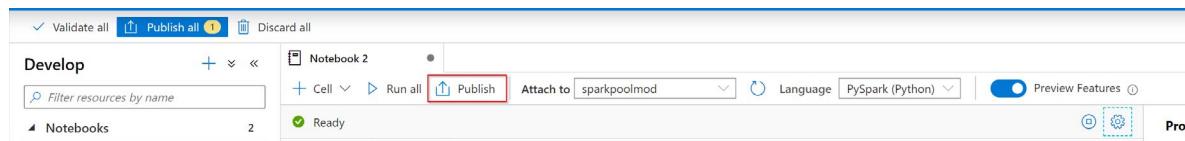
```
Cell 3
1 from pyspark.sql import SparkSession
2 from pyspark.sql.types import *
3 from pyspark.sql.functions import *
4
5 profitByDateProduct = (data_path.groupBy("TransactionDate", "ProductId")
6     .agg(
7         sum("ProfitAmount").alias("(sum)ProfitAmount"),
8         round(avg("Quantity"), 4).alias("(avg)Quantity"),
9         sum("Quantity").alias("(sum)Quantity"))
10    .orderBy("TransactionDate"))
11 display(profitByDateProduct.limit(100))

Command executed in 6s 545ms by joel on 09-10-2020 15:13:18.982 -04:00
> Job execution Succeeded  Spark 3 executors 12 cores
View in monitoring  Open Spark UI ▾
View Table Chart
TransactionDate ProductId (sum)ProfitAmount (avg)Quantity (sum)Quantity
20101231 64 52975.23 2.5547 5467
20101231 3348 1409.01 2.4512 201
20101231 1470 1595.58 2.6364 174
20101231 1266 10731.05 2.5896 1199
20101231 48 39516.75 2.5158 4053
20101231 132 42468.96 2.4995 5154
20101231 66 11255.79 2.4026 1007
Ready (Stop session) | Configure session
```

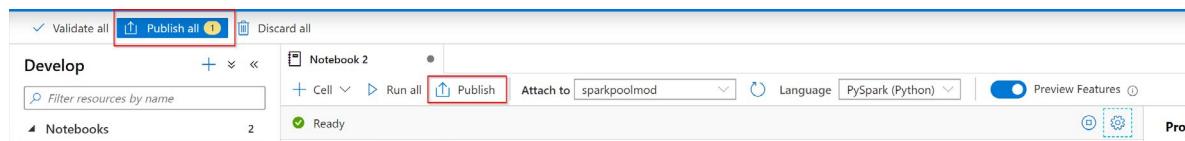
Save spark notebooks

When you have finished with your work in the notebooks, it is possible to save a single notebook or all of notebooks that you've created within Azure Synapse Studio notebooks.

- To save changes you made to a single notebook, select the **Publish** button on the notebook command bar.



- To save all notebooks in your workspace, select the **Publish all** button on the workspace command bar.



In the notebook properties, you can configure whether to include the cell output when saving.

The screenshot shows a 'Properties' dialog box for a notebook. At the top right are three icons: a blue gear (Settings), a red square with a white gear (Notebook settings), and three dots (More). The 'General' tab is selected. A callout box highlights the 'Notebook settings' icon. Inside the dialog, there is a note: 'Choose a name for your Notebook. This name can be updated at any time until it is published.' The 'Name' field contains 'Notebook 2'. The 'Description' field is empty. Under 'Type', it says '.ipynb notebook'. Under 'Size', it says '1,386 bytes'. A section titled 'Notebook settings' contains a checked checkbox for 'Include cell output when saving'. Below this is a 'Session' section with a 'Configure session' link.

Properties

General

Notebook settings

Include cell output when saving

Name
Notebook 2

Description

Type
.ipynb notebook

Size
1,386 bytes

Session
[Configure session](#)

Knowledge check

Question 1

What are Azure Synapse Studio notebooks based on?

- dedicated SQL pool.
- Apache Spark.
- Apache Spark pool.

Question 2

How can all notebooks in Synapse studio be saved?

- Select the **Publish all** button on the workspace command bar.
- Select the **Publish** button on the notebook command bar.
- Using CTRL + S.

Summary

In this module, you have learned how to ingest data using Apache Spark notebooks in Azure Synapse Analytics.

Now that you have completed this module, you are able to:

- Understand Apache Spark notebooks
- Understand the use-cases for Apache Spark notebooks
- Create an Apache Spark notebook in Azure Synapse Analytics
- Discover supported languages in Apache Spark notebooks
- Develop Apache Spark notebooks
- Run Apache Spark notebooks
- Load data in Apache Spark notebooks
- Save Apache Spark notebooks

Transform data with DataFrames in Apache Spark Pools in Azure Synapse Analytics

Introduction

In this module, you will learn how to transform data using DataFrames in Apache Spark pools in Azure Synapse Analytics.

After completing this module, you will be able to:

- Describe DataFrames in Apache Spark pools in Azure Synapse Analytics
- Load data into an Apache Spark DataFrame
- Create an Apache Spark table
- Write data to and from a storage account
- Flatten nested structures and explode arrays with Apache Spark

Prerequisites

Before taking this module, it is recommended that you complete the following modules:

- Azure Data Fundamentals
- Introduction to Azure Data Factory
- Introduction to Azure Synapse Analytics

Introduction to dataframes in spark pools in Azure Synapse Analytics

DataFrames are a collection of data organized into named columns. DataFrames enable Apache Spark to understand the schema of the data, and optimize any execution plans on queries that will access the data held in the DataFrame. DataFrames are designed to process a large volume of data from a wide variety of data sources from structured data sources through to Resilient Distributed Datasets (RDDs) in either a batch or streaming data architecture. In short, DataFrames are to Apache Spark, what tables are to relational databases.

The first step is to construct a DataFrame. You can create a DataFrame and populate it with data at execution time as shown in the following example:

```
new_rows = [('CA',22, 45000),("WA",35,65000) , ("WA",50,85000)]
demo_df = spark.createDataFrame(new_rows, ['state', 'age', 'salary'])
demo_df.show()
```

In the example above, the variable named **new_rows** creates the data in the code segment to store in the DataFrame. Then a second variable named **demo_df** is created to use the `spark.createDataFrame` method, which refers to the **new_rows** variable in the first parameter, and in the second parameter defines the column heading names for the DataFrame as state, age, and salary. The third line uses the `show` method to output the results of the **demo_df** variable.

However, it is more common to ingest data from a data source such as a file into a DataFrame as shown in the next example:

```
from azureml.opendatasets import NyCTlcYellow  
  
data = NyCTlcYellow()  
data_df = data.to_spark_dataframe()  
display(data_df.limit(10))
```

In this example, the New York Taxi open-source data is imported and stored in a variable named data. The second line creates an Apache Spark DataFrame in a variable named data_df using the to_spark_dataframe() method. Finally, 10 rows of data are returned back from the data_df variable using the display method.

Once you're at the stage where you have populated a DataFrame with data, you manipulate the data stored in a DataFrame. The manipulation of data can be done with User Defined Functions (UDFs) that are column-based and help you transform and manipulate the data stored in a DataFrame.

Load data into a spark dataframe

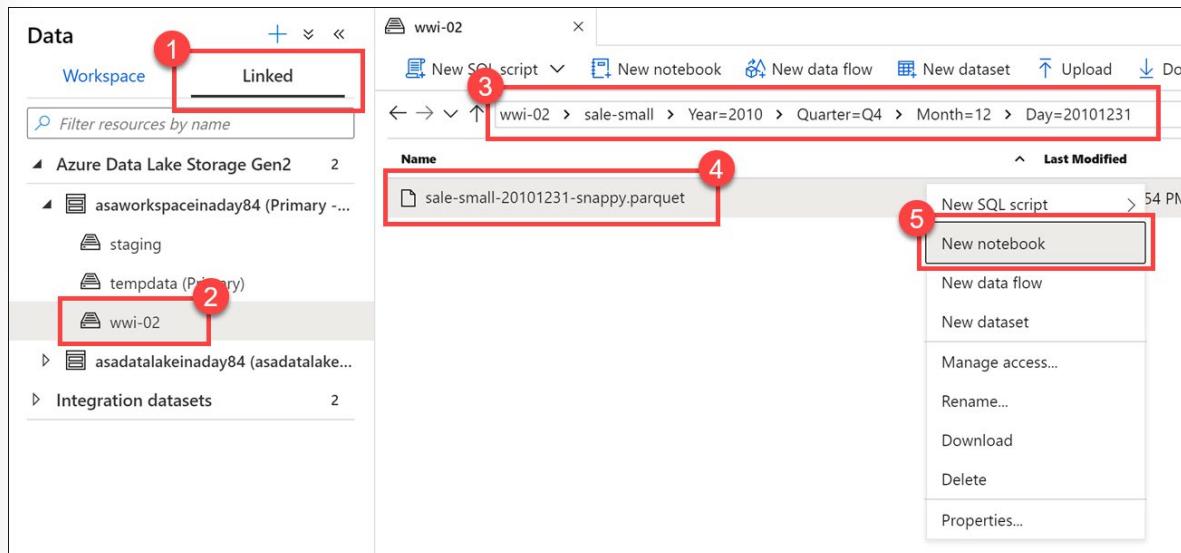
You can load data into an Apache Spark DataFrame from different file types stored in an Azure Storage Account, or from data stored in a dedicated SQL pool.

Some examples of loading data are:

- Read a CSV from Azure Data Lake Store Gen2 as an Apache Spark DataFrame
- Read a CSV from Azure Storage Account as an Apache Spark DataFrame
- Read data from the primary storage account

Let's take an example of the company Tailwind Traders. Tailwind Traders has Parquet files stored in their data lake. They want to know how they can quickly access the files and explore them using Apache Spark.

One option to create a DataFrame is by using the Data hub in Azure Synapse Studio to view the Parquet files in a connected storage account. It is achieved by using the *new notebook* context menu to create a new Azure Synapse Notebook that loads a Spark DataFrame with the contents of a selected parquet file.



A notebook with the associated PySpark code is generated to load the data into an Apache Spark DataFrame and display rows with the header. It also automatically creates the connection to the storage account and file in the data_path section.



If you would like to load data to or from a table into a Spark DataFrame, you can use the Azure Synapse Apache Spark pool to Synapse SQL connector. The Azure Synapse Apache Spark pool to Synapse SQL connector is a data source implementation for Apache Spark, and it uses Azure Data Lake Storage Gen2 and PolyBase in dedicated SQL pools to efficiently transfer data between the Spark cluster and the Azure Synapse dedicated SQL pool instance.

So let's say you want to load the NYC Taxi data into the Spark database named nyctaxi, and assume the data is available in a table stored in SQLPOOL1.

How can you load it into an Apache Spark database named nyctaxi?

You would perform the following steps:

1. In Azure Synapse Studio, go to the **Develop** hub.
2. Select **+**, and then select **Notebook**.
3. On the top of the notebook, set **Attach** to the value of **Spark1**.
4. Select **Add code** to add a notebook code cell, and then paste the following text:

```
Copy
%%spark
spark.sql("CREATE DATABASE IF NOT EXISTS nyctaxi")
val df = spark.read.sqlanalytics("SQLPOOL1.dbo.Trip")
df.write.mode("overwrite").saveAsTable("nyctaxi.trip")
```

In this code example, the `spark.sql` method is used to create a database named `nyctaxi`. A `DataFrame` named `df` reads data from a table named `Trip` in the `SQLPOOL1` dedicated SQL pool instance. Finally, the `DataFrame` `df` writes data into it and used the `saveAsTable` method to save it as `nyctaxi.trip`.

As you can see, there are various ways to load data into an Apache Spark DataFrame depending on the source.

Exercise: Load data into a spark dataframe

You have customer profile data from an e-commerce system that provides top product purchases for each visitor of the site (customer) over the past 12 months. This data is stored within JSON files in the data lake. They have struggled with ingesting, exploring, and transforming these JSON files and want your guidance. The files have a hierarchical structure that they want to flatten before loading into relational data stores. They also wish to apply grouping and aggregate operations as part of the data engineering process.

You recommend using Azure Synapse notebooks to explore and apply data transformations on the JSON files.

This exercise will focus on how to load data into an Apache Spark DataFrame.

1. Create a new cell in the Apache Spark notebook, add the following code beneath the code in the cell to define a variable named `datalake` whose value is the name of the primary storage account (replace the `REPLACE_WITH_YOUR_DATALAKE_NAME` value with the name of the storage account):

```
datalake = 'REPLACE_WITH_YOUR_DATALAKE_NAME'
```

2. Create a new cell in the Apache Spark notebook, enter the code below and execute the cell:

```
df = (spark.read \
    .option('inferSchema', 'true') \
    .json('abfss://wwi-02@' + datalake + '.dfs.core.windows.net/online-user-profiles-02/*.json', \
    multiLine=True)
)

df.printSchema()
```

The `datalake` variable we created in the first cell is used here as part of the file path.

Your output should look like as follows:

```
root
|-- topProductPurchases: array (nullable = true)
|   |-- element: struct (containsNull = true)
|       |-- itemsPurchasedLast12Months: long (nullable = true)
|       |-- productId: long (nullable = true)
|-- visitorId: long (nullable = true)
```

Notice that we are selecting all JSON files within the `online-user-profiles-02` directory. Each JSON file contains several rows, which is why we specified the `multiLine=True` option. Also, we set the `inferSchema` option to `true`, which instructs the Apache Spark engine to review the files and create a schema based on the nature of the data.

3. We have been using Python code in these cells up to this point. If we want to query the files using SQL syntax, one option is to create a temporary view of the data within the DataFrame. Execute the code below in a new cell to create a view named `user_profiles`:

```
# create a view called user_profiles
df.createOrReplaceTempView("user_profiles")
```

4. Create a new cell. Since we want to use SQL instead of Python, we use the `%%sql` magic to set the language of the cell to SQL. Execute the below code in the cell:

```
%%sql
```

```
SELECT * FROM user_profiles LIMIT 10
```

Notice that the output shows nested data for `topProductPurchases`, which includes an array of `productId` and `itemsPurchasedLast12Months` values. You can expand the fields by clicking the right triangle in each row.

The screenshot shows a Jupyter Notebook cell with the following content:

```
1 %%sql
2
3 SELECT * FROM user_profiles LIMIT 10
```

Command executed in 12s 994ms by joel on 09-10-2020 15:50:03.604 -04:00

> Job execution Succeeded Spark 3 executors 12 cores

View Table Chart

topProductPurchases visitorId

topProductPurchases	visitorId
► [{"schema": [{"name": "itemsPurcha"}]	117000
► [{"schema": [{"name": "itemsPurcha"}]	117001
► ► [{"schema": [{"name": "itemsPurcha"}]	117002
► ► 0: {"schema": [{"name": "itemsPurcha"}]	
► ► 1: {"schema": [{"name": "itemsPurcha"}]	
► ► schema: [{"name": "itemsPurcha"}]	
► ► 0: {"name": "itemsPurchasedLast12Mo	
► ► name: "itemsPurchasedLast12Mo	
► ► dataType: "0"	
► ► nullable: "true"	
► ► metadata: {"map": {}}	
► ► map: "0"	
► ► 1: {"name": "productId", "dat	
► ► values: "[15,2515]"	
► ► 2: {"schema": [{"name": "itemsPurcha"}]	
► ► 3: {"schema": [{"name": "itemsPurcha"}]	
► ► [{"schema": [{"name": "itemsPurcha"}]	117003

The JSON nested output makes analyzing the data a bit difficult. It is because the JSON file contents looks as follows:

```
[  
{  
  "visitorId": 9529082,  
  "topProductPurchases": [  
    {  
      "productId": 4679,  
      "itemsPurchasedLast12Months": 26  
    },  
    {  
      "productId": 1779,  
      "itemsPurchasedLast12Months": 32  
    },  
    {  
      "productId": 2125,  
      "itemsPurchasedLast12Months": 75  
    }  
  ]  
}
```

```
"productId": 2007,  
"itemsPurchasedLast12Months": 39  
},  
{  
    "productId": 1240,  
    "itemsPurchasedLast12Months": 31  
},  
{  
    "productId": 446,  
    "itemsPurchasedLast12Months": 39  
},  
{  
    "productId": 3110,  
    "itemsPurchasedLast12Months": 40  
},  
{  
    "productId": 52,  
    "itemsPurchasedLast12Months": 2  
},  
{  
    "productId": 978,  
    "itemsPurchasedLast12Months": 81  
},  
{  
    "productId": 1219,  
    "itemsPurchasedLast12Months": 56  
},  
{  
    "productId": 2982,  
    "itemsPurchasedLast12Months": 59  
}  
]  
},  
{  
    ...  
},  
{  
    ...  
}  
]
```

Exercise: Create a spark table

In this exercise, you will create an Apache Spark table

1. The Apache Spark engine can analyze the Parquet files and infer the schema. To do this analysis, enter the below code in the new cell and **run** it:

```
data_path.printSchema()
```

Your output should look as follows:

```
root
|-- TransactionId: string (nullable = true)
|-- CustomerId: integer (nullable = true)
|-- ProductId: short (nullable = true)
|-- Quantity: short (nullable = true)
|-- Price: decimal(29,2) (nullable = true)
|-- TotalAmount: decimal(29,2) (nullable = true)
|-- TransactionDate: integer (nullable = true)
|-- ProfitAmount: decimal(29,2) (nullable = true)
|-- Hour: byte (nullable = true)
|-- Minute: byte (nullable = true)
|-- StoreId: short (nullable = true)
```

Apache Spark evaluates the file contents to infer the schema. This automatic inference is sufficient for data exploration and most transformation tasks. However, when you load data to an external resource like a SQL pool table, sometimes you need to declare your own schema and apply that to the dataset. For now, the schema looks good.

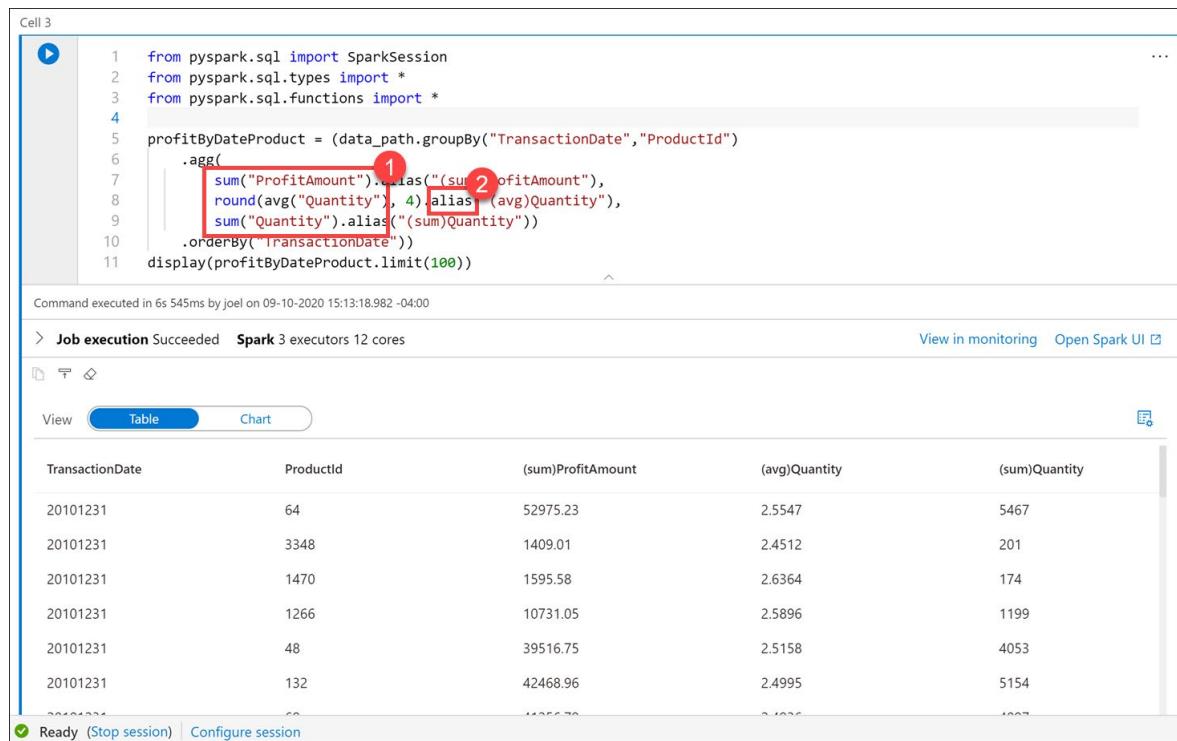
2. Now let's use the DataFrame to use aggregates and grouping operations to better understand the data. Create a new cell and enter the following, then **run** the cell:

```
from pyspark.sql import SparkSession
from pyspark.sql.types import *
from pyspark.sql.functions import *

profitByDateProduct = (data_path.groupBy("TransactionDate", "ProductId")
    .agg(
        sum("ProfitAmount").alias("(sum)ProfitAmount"),
        round(avg("Quantity"), 4).alias("(avg)Quantity"),
        sum("Quantity").alias("(sum)Quantity"))
    .orderBy("TransactionDate"))
display(profitByDateProduct.limit(100))
```

We import required Python libraries to use aggregation functions and types defined in the schema to successfully execute the query.

The output shows the same data we saw in the chart above, but now with `sum` and `avg` aggregates **(1)**. Notice that we use the `alias` method **(2)** to change the column names.



```
Cell 3
1 from pyspark.sql import SparkSession
2 from pyspark.sql.types import *
3 from pyspark.sql.functions import *
4
5 profitByDateProduct = (data_path.groupBy("TransactionDate","ProductId")
6     .agg(
7         sum("ProfitAmount").alias("(sum)ProfitAmount"),
8         round(avg("Quantity"), 4).alias("(avg)Quantity"),
9         sum("Quantity").alias("(sum)Quantity"))
10    .orderBy("TransactionDate"))
11 display(profitByDateProduct.limit(100))

Command executed in 6s 545ms by joel on 09-10-2020 15:13:18.982 -04:00
> Job execution Succeeded  Spark 3 executors 12 cores
View in monitoring  Open Spark UI ▾
View Table Chart
TransactionDate ProductId (sum)ProfitAmount (avg)Quantity (sum)Quantity
20101231 64 52975.23 2.5547 5467
20101231 3348 1409.01 2.4512 201
20101231 1470 1595.58 2.6364 174
20101231 1266 10731.05 2.5896 1199
20101231 48 39516.75 2.5158 4053
20101231 132 42468.96 2.4995 5154
20101231 66 14255.70 2.4820 1927
Ready (Stop session) | Configure session
```

Flatten nested structures and explode arrays with Apache Spark

A common use case for using Apache Spark pools in Azure Synapse Analytics is for transforming complex data structures using DataFrames. It can help for the following reasons:

- Complex data types are increasingly common and represent a challenge for data engineers because analyzing nested schema and data arrays often include time-consuming and complex SQL queries.
- It can be difficult to rename or cast the nested column data type.
- Performance issues arise when working with deeply nested objects. Data Engineers need to understand how to efficiently process complex data types and make them easily accessible to everyone. In the following example, Apache Spark for Azure Synapse is used to read and transform objects into a flat structure through data frames.
- Apache Synapse SQL serverless is used to query such objects directly and return those results as a regular table.
- With Azure Synapse Apache Spark pools, it's easy to transform nested structures into columns and array elements into multiple rows.

In the example, the following steps show the techniques involved to deal with complex data types by creating multiple DataFrames to achieve the desired result.



Step 1: Define a function for flattening

We create a function that will flatten the nested schema.

Step 2: Flatten nested schema

We will define the function you create to flatten the nested schema from one DataFrame into a new DataFrame.

Step 3: Explode Arrays

Here you will transform the data array from the DataFrame created in step 2 into a new DataFrame.

Step 4: Flatten child nested Schema

Finally, you use the transformed DataFrame created in step 3 and load the cleansed data into a destination DataFrame to complete the work.

Exercise: Flatten nested structures and explode arrays with Apache Spark in synapse

In this exercise, you will learn how to work with complex data structure and use functions to view data more easily

1. PySpark contains a special **explode function⁴**, which returns a new row for each element of the array. The new row will help flatten the `topProductPurchases` column for better readability or for easier querying. Execute the code below in a new cell:

```
from pyspark.sql.functions import udf, explode
```

```
flat=df.select('visitorId',explode('topProductPurchases').alias('topProductPurchases_flat'))  
flat.show(100)
```

In this cell, we created a new DataFrame named `flat` that includes the `visitorId` field and a new aliased field named `topProductPurchases_flat`. As you can see, the output is a bit easier to read and, by extension, easier to query.

⁴ <https://spark.apache.org/docs/latest/api/python/pyspark.sql.html?highlight=explode#pyspark.sql.functions.explode>

Cell 7

```
1 from pyspark.sql.functions import udf, explode
2
3 flat=df.select('visitorId',explode('topProductPurchases').alias('topProductPurchases_flat'))
4 flat.show(100)
```

Command executed in 2s 400ms by joel on 09-10-2020 15:55:17.768 -04:00

> Job execution Succeeded Spark 3 executors 12 cores

visitorId	topProductPurchases_flat
117000	[13, 3623]
117000	[5, 2321]
117001	[93, 713]
117001	[19, 2144]
117001	[30, 1094]
117001	[82, 3223]
117001	[42, 3328]
117001	[62, 2926]
117001	[63, 2651]
117001	[39, 341]
117001	[85, 4841]
117001	[67, 4289]
117001	[42, 1264]
117001	[43, 3608]
117001	[14, 504]
117001	[97, 2649]
117001	[44, 2873]
117001	[7, 4491]

2. Create a new cell and execute the following code to create a new flattened version of the DataFrame that extracts the `topProductPurchases_flat.productId` and `topProductPurchases_flat.itemsPurchasedLast12Months` fields to create new rows for each data combination:

```
topPurchases = (flat.select('visitorId','topProductPurchases_flat.productId','topProductPurchases_flat.itemsPurchasedLast12Months')
    .orderBy('visitorId'))
```

```
topPurchases.show(100)
```

In the output, notice that we now have multiple rows for each `visitorId`.

Cell 8

```
1 topPurchases = (flat.select('visitorId','topProductPurchases_flat.productId','topProductPurchases_flat.itemsPurchasedLast12Months')
2 | .orderBy('visitorId'))
3
4 topPurchases.show(100)
```

Command executed in 3s 712ms by joel on 09-10-2020 15:59:40.419 -04:00

> Job execution Succeeded Spark 3 executors 12 cores

View in monitoring Open Spark UI

visitorId	productId	itemsPurchasedLast12Months
80000	4198	92
80000	2488	31
80000	4136	30
80000	1362	18
80000	3122	33
80000	3270	5
80000	93	86
80000	102	42
80000	4206	21
80000	3538	65
80000	4745	38
80000	291	49
80000	290	83
80000	4074	48
80000	4024	35
80000	2481	63
80000	2859	54
80000	2069	93
80000	1330	78
80000	3794	90
80001	4105	11
80001	2249	75
80001	4684	9
80001	3729	56

3. Let's order the rows by the number of items purchased in the last 12 months. Create a new cell and execute the following code:

```
# Let's order by the number of items purchased in the last 12 months
sortedTopPurchases = topPurchases.orderBy("itemsPurchasedLast12Months")

display(sortedTopPurchases.limit(100))
```

Cell 9

```
1 # Let's order by the number of items purchased in the last 12 months
2 sortedTopPurchases = topPurchases.orderBy("itemsPurchasedLast12Months")
3
4 display(sortedTopPurchases.limit(100))
```

Command executed in 6s 13ms by joel on 09-10-2020 16:02:31.831 -04:00

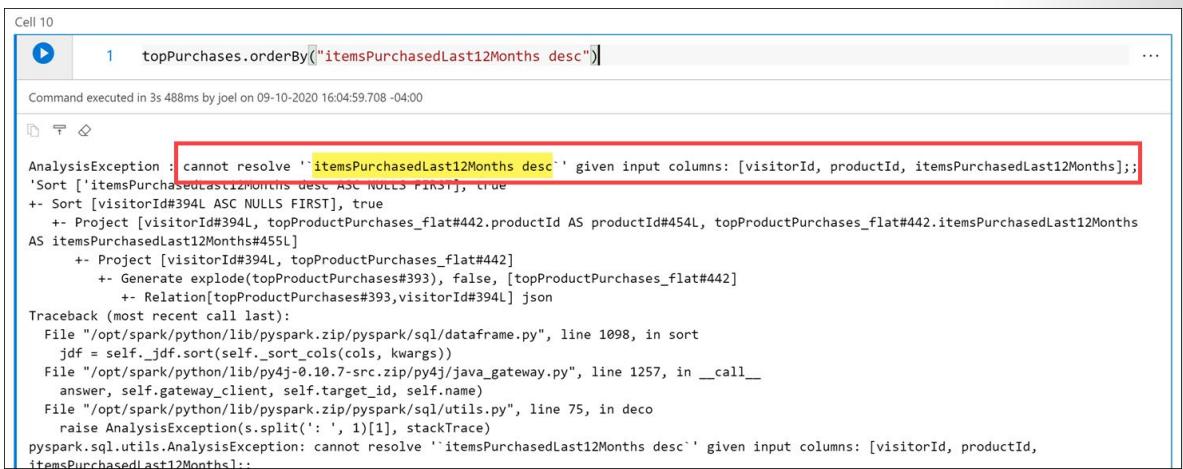
> **Job execution** Succeeded **Spark** 3 executors 12 cores

View **Table** **Chart**

visitorId	productId	itemsPurchasedLast12Months
118878	2895	1
88702	3331	1
118888	4497	1
118027	3405	1
118900	4338	1
118068	661	1
118900	4062	1
118088	4394	1
118906	226	1
118112	3509	1
118017	601	1

4. How do we sort in reverse order? You might conclude that we could make a call like this: `topPurchases.orderBy("itemsPurchasedLast12Months desc")`. Try it in a new cell:

```
topPurchases.orderBy("itemsPurchasedLast12Months desc")
```



```
Cell 10
1 topPurchases.orderBy("itemsPurchasedLast12Months desc")
...
Command executed in 3s 488ms by joel on 09-10-2020 16:04:59.708 -04:00
...
AnalysisException : cannot resolve 'itemsPurchasedLast12Months desc' given input columns: [visitorId, productId, itemsPurchasedLast12Months];
'Sort [itemsPurchasedLast12Months desc ASC NULLS FIRST], true
+- Sort [visitorId#394L ASC NULLS FIRST], true
  +- Project [visitorId#394L, topProductPurchases_flat#442.productId AS productId#454L, topProductPurchases_flat#442.itemsPurchasedLast12Months
AS itemsPurchasedLast12Months#455L]
    +- Project [visitorId#394L, topProductPurchases_flat#442]
      +- Generate explode(topProductPurchases#393), false, [topProductPurchases_flat#442]
        +- Relation[topProductPurchases#393,visitorId#394L] json
Traceback (most recent call last):
File "/opt/spark/python/lib/pyspark.zip/pyspark/sql/dataframe.py", line 1098, in sort
    jdf = self._jdf.sort(self._sort_cols(cols, kwargs))
File "/opt/spark/python/lib/py4j-0.10.7-src.zip/py4j/java_gateway.py", line 1257, in __call__
    answer, self.gateway_client, self.target_id, self.name)
File "/opt/spark/python/lib/pyspark.zip/pyspark/sql/utils.py", line 75, in deco
    raise AnalysisException(s.split(': ', 1)[1], stackTrace)
pyspark.sql.utils.AnalysisException: cannot resolve 'itemsPurchasedLast12Months desc' given input columns: [visitorId, productId,
itemsPurchasedLast12Months].
```

Notice that there is an **AnalysisException** error, because `itemsPurchasedLast12Months desc` does not match up with a column name.

5. The **Column** class is an object that encompasses more than just the name of the column, but also column-level-transformations, such as sorting in a descending order. Execute the following code in a new cell:

```
sortedTopPurchases = (topPurchases
    .orderBy( col("itemsPurchasedLast12Months").desc() ))

display(sortedTopPurchases.limit(100))
```

Notice that the results are now sorted by the `itemsPurchasedLast12Months` column in descending order, thanks to the `desc()` method on the `col` object.

Cell 11

```
1 sortedTopPurchases = (topPurchases
2     .orderBy( col("itemsPurchasedLast12Months").desc() ))
3
4 display(sortedTopPurchases.limit(100))
```

Command executed in 5s 827ms by joel on 09-10-2020 16:10:15.556 -04:00

> Job execution Succeeded Spark 3 executors 12 cores

View Table Chart

visitorId	productId	itemsPurchasedLast12Months
84884	2834	99
101990	835	99
84902	1482	99
84011	340	99
84906	139	99
84024	3876	99
84915	4748	99
84060	484	99
84934	1359	99
84066	4467	99

6. How many *types* of products did each customer purchase? To find the answer, we need to group by `visitorId` and aggregate on the number of rows per customer. Execute the following code in a new cell:

```
groupedTopPurchases = (sortedTopPurchases.select("visitorId")
    .groupBy("visitorId")
    .agg(count("*").alias("total"))
    .orderBy("visitorId") )
```

```
display(groupedTopPurchases.limit(100))
```

Notice how we use the `groupBy` method on the `visitorId` column, and the `agg` method over a count of records to display the total for each customer.

Cell 12

```
1 groupedTopPurchases = (sortedTopPurchases.select("visitorId")
2     .groupBy("visitorId")
3     .agg(count("*").alias("total"))
4     .orderBy("visitorId") )
5
6 display(groupedTopPurchases.limit(100))
```

Command executed in 5s 21ms by joel on 09-10-2020 16:16:23.042 -04:00

> Job execution Succeeded Spark 3 executors 12 cores

View Table Chart

visitorId	total
80000	20
80001	20
80002	15
80003	12
80004	10
80005	13
80006	6
80007	18
80008	4

7. How many *total items* did each customer purchase? To find the answer, we need to group by `visitorId` and aggregate on the sum of `itemsPurchasedLast12Months` values per customer. Execute the following code in a new cell:

```
groupedTopPurchases = (sortedTopPurchases.select("visitorId","itemsPurchasedLast12Months")
    .groupBy("visitorId")
    .agg(sum("itemsPurchasedLast12Months").alias("totalItemsPurchased"))
    .orderBy("visitorId") )
```

```
groupedTopPurchases.show(100)
```

Here we group by `visitorId` once again, but now we use a `sum` over the `itemsPurchasedLast12Months` column in the `agg` method. Notice that we included the `itemsPurchasedLast12Months` column in the `select` statement so we could use it in the `sum`.

Cell 13

```
1 groupedTopPurchases = (sortedTopPurchases.select("visitorId", "itemsPurchasedLast12Months")
2     .groupBy("visitorId")
3     .agg(sum("itemsPurchasedLast12Months").alias("totalItemsPurchased"))
4     .orderBy("visitorId") )
5
6 display(groupedTopPurchases.limit(100))
```

Command executed in 5s 227ms by joel on 09-10-2020 16:21:05.194 -04:00

> Job execution Succeeded Spark 3 executors 12 cores

View Table Chart

visitorId	totalItemsPurchased
80000	1054
80001	834
80002	754
80003	684
80004	598
80005	615
80006	348
80007	932
80008	199

Knowledge check

Question 1

What is a step in flattening a nested schema?

- Create a parquet file.
- Explode Arrays.
- Load a csv file.

Question 2

What is a dataframe?

- A creation of a data structure.
- A parquet file.
- A csv file.

Summary

In this module, you have learned how to transform data using DataFrames in Apache Spark pools in Azure Synapse Analytics.

Now that you have completed this module, you are able to:

- Describe DataFrames in spark pools in Azure Synapse Analytics
- Load data into an Apache Spark DataFrame
- Create an Apache Spark table
- Write data to and from a storage account
- Flatten nested structures and explode arrays with Apache Spark

Integrate SQL and Apache Spark pools in Azure Synapse Analytics

Introduction

In this module, you will learn how to integrate SQL pools and Apache Spark pools in Azure Synapse Analytics.

After completing this module, you will be able to:

- Describe the integration methods between SQL and Apache Spark pools in Azure Synapse Analytics
- Understand the use cases for SQL and Apache Spark pools integration
- Authenticate in Azure Synapse Analytics
- Transfer data between SQL and Apache Spark pools in Azure Synapse Analytics
- Authenticate between Apache Spark and SQL pools in Azure Synapse Analytics
- Integrate SQL and Apache Spark pools in Azure Synapse Analytics
- Externalize the use of Apache Spark pools within Azure Synapse Workspace
- Transfer data outside the Synapse workspace using the PySpark connector

Prerequisite

Before taking this module, it is recommended that you complete the following modules:

- Azure Data Fundamentals
- Introduction to Azure Data Factory
- Introduction to Azure Synapse Analytics

Describe the integration methods between SQL and spark pools in Azure Synapse Analytics

By creating analytical solutions within Azure Synapse Analytics, the need for setting up multiple services to connect an Apache Spark cluster to a SQL database is no longer required. The Azure Synapse Analytics environment enables you to use both technologies within one integrated platform. The integrated platform experience allows you to switch between Apache Spark and SQL based data engineering tasks applicable to the expertise you have in-house. As a result, an Apache Spark-orientated data engineer can easily communicate and work with a SQL-based data engineer on the same platform.

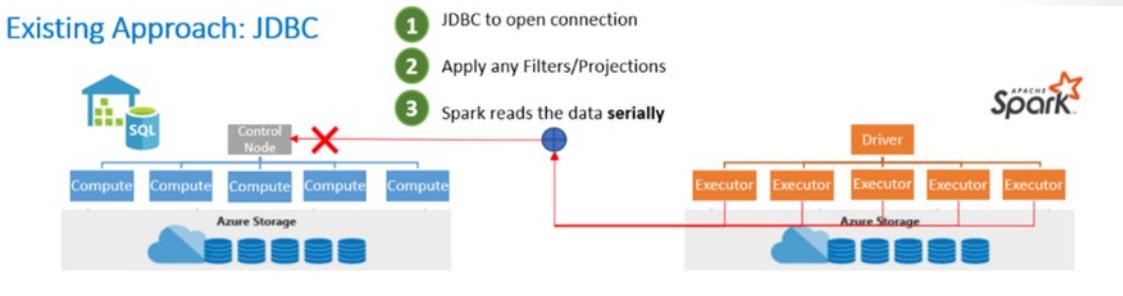
The interoperability between Apache Spark and SQL helps you achieve as follows:

- Work with SQL and Apache Spark to directly explore and analyze Parquet, CSV, TSV, and JSON files stored in the data lake.
- Enable fast, scalable loads for data transferring between SQL and Apache Spark databases.
- Make use of a shared Hive-compatible metadata system that enables you to define tables on files in the data lake such that it can be consumed by either Apache Spark or Hive.

It raises the question of how SQL and Apache Spark integration works. That's where the Azure Synapse Apache Spark to Synapse SQL connector comes in to play.

The Azure Synapse Apache Spark to Synapse SQL connector is designed to efficiently transfer data between serverless Apache Spark pools and dedicated SQL pools in Azure Synapse. At the moment, the Azure Synapse Apache Spark to Synapse SQL connector works on dedicated SQL pools only, it doesn't work with serverless SQL pools.

In the existing approach, you often see the use of the Java DataBase Connectivity (JDBC) Application Programming Interface (API). The JDBC API opens the connection, filters, and applies projections, and Apache Spark reads the data serially. Given that two distributed systems such as Apache Spark and SQL pools are being used, using the JDBC API becomes a bottleneck with serial transfer of data.



Therefore, a new approach is to use both JDBC and PolyBase. First, the JDBC opens a connection and issues Create External Tables As Select (CETAS) statements and sends filters and projections. The filters and projections are then applied to the data warehouse and exports the data in parallel using PolyBase. Apache Spark reads the data in parallel based on the user-provisioned workspace and the default data lake storage.



As a result, you can use the Azure Synapse Apache Spark Pool to Synapse SQL connector to transfer data between a Data Lake store via Apache Spark and dedicated SQL Pools efficiently.

Understand the use-cases for SQL and spark pools integration

The Apache Spark and SQL integration that is available within Azure Synapse analytics provides several benefits:

- You can take advantage of the big data computational power that Apache Spark offers
- There is flexibility in the use of Apache Spark and SQL languages and frameworks on one platform
- The integration is seamless and doesn't require a complex setup
- SQL and Apache Spark share the same underlying metadata store to transfer data easily

As a result, when you deploy an Azure Synapse Apache Spark cluster, the Azure Data Lake Gen2 capability enables you to store Apache Spark SQL Tables within it. If you use Apache Spark SQL tables, these tables can be queried from a SQL-based Transact-SQL language without needing to use commands like

CREATE EXTERNAL TABLE. Within Azure Synapse Analytics, these queries integrate natively with data files that are stored in an Apache Parquet format.

The integration can be helpful in use cases where you perform an ETL process predominately using SQL but need to call on the computation power of Apache Spark to perform a portion of the extract, transform, and load (ETL) process as it is more efficient.

Let's say you would like to write data to a SQL pool after you've performed engineering tasks in Apache Spark. You can reference the dedicated SQL pool data as a source for joining with an Apache Spark DataFrame that can contain data from other files. The method makes use of the Azure Synapse Apache Spark to Synapse SQL connector to efficiently transfer data between the Apache Spark and SQL Pools.

The Azure Synapse Apache Spark pool to Synapse SQL connector is a data source implementation for Apache Spark. It uses the Azure Data Lake Storage Gen2 and PolyBase in SQL pools to efficiently transfer data between the Spark cluster and the Synapse SQL instance.

The other thing to keep in mind is that beyond the capabilities mentioned above, the Azure Synapse Studio experience gives you an integrated notebook experience. Within this notebook experience, you can attach a SQL or Apache Spark pool, and develop and execute transformation pipelines using Python, Scala, and native Spark SQL.

Authenticate in Azure Synapse Analytics

Now that you understand the integration methods between SQL and Apache Spark pools, and hopefully understand the use case it works with, it is imperative to understand how the services authenticate within Azure Synapse Analytics. The authentication between the two systems is made seamless in Azure Synapse Analytics. The Token Service connects with Azure Active Directory to obtain the security tokens to be used when accessing the storage account or the data warehouse in the dedicated SQL pool.

For this reason, there's no need to create credentials or specify them in the connector API if Azure AD-Auth is configured at the storage account and the dedicated SQL pool. If not, SQL Authentication can be specified. The only constraint is that this connector only works in Scala.

There are some prerequisites to authenticate correctly:

- The account used needs to be a member of db_exporter role in the database or SQL pool from which you transfer data to or from.
- The account used needs to be a member of the Storage Blob Data Contributor role on the default storage account.

If you want to create users, you need to connect to the dedicated SQL pool database from which you want transfer data to or from as shown in the following example:

```
--SQL Authenticated User
CREATE USER Leo FROM LOGIN Leo;

--Azure Active Directory User
CREATE USER [chuck@contoso.com] FROM EXTERNAL PROVIDER;
```

When you want to assign the user account to a role, you can use the following script as an example:

```
--SQL Authenticated user
EXEC sp_addrolemember 'db_exporter', 'Leo';
--Azure Active Directory User
EXEC sp_addrolemember 'db_exporter', [chuck@contoso.com]
```

Once the authentication is in place, you can transfer data to or from a dedicated SQL pool attached within the workspace.

Transfer data between SQL and spark pool in Azure Synapse Analytics

In this section, you will learn about using Azure Active Directory to transfer data to and from an Apache Spark pool and a dedicated SQL pool attached within the workspace you have created for your Azure Synapse Analytics account. If you're using the notebook experience from the Azure Synapse Studio environment linked to your workspace resource, you don't have to use import statements. Import statements are only required when you don't go through the integrated notebook experience.

It is important that the Constants and the SqlAnalyticsConnector are set up as shown below:

```
#scala  
import com.microsoft.spark.sqlanalytics.utils.Constants  
import org.apache.spark.sql.SqlAnalyticsConnector._
```

To read data from a dedicated SQL pool, you should use the Read API. The Read API works for Internal tables (Managed Tables) and External Tables in the dedicated SQL pool.

The Read API using Azure AD looks as follows:

```
#scala  
val df = spark.read.sqlanalytics("<DBName>.<Schema>.<TableName>")
```

The parameters it takes in are:

- **DBName:** the name of the database.
- **Schema:** the schema definition such as dbo.
- **TableName:** the name of the table you want to read data from.

To write data to a dedicated SQL Pool, you should use the Write API. The Write API creates a table in the dedicated SQL pool. Then, it invokes Polybase to load the data into the table that was created. One thing to keep in mind is that the table can't already exist in the dedicated SQL pool. If that happens, you'll receive an error stating: "There is already an object named..."

The Write API using Azure Active Directory (Azure AD) looks as follows:

```
df.write.sqlanalytics("<DBName>.<Schema>.<TableName>", <TableType>)
```

The parameters it takes in are:

- **DBName:** the name of the database.
- **Schema:** the schema definition such as dbo.
- **TableName:** the name of the table you want to read data from.
- **TableType:** specification of the type of table, which can have two values.
 - Constants.INTERNAL - Managed table in dedicated SQL pool
 - Constants.EXTERNAL - External table in dedicated SQL pool

The TableType parameter in the Write API has some extra parameters to consider as mentioned above.

An example of a SQL pool-managed table looks as follows:

```
df.write.sqlAnalytics("<DBName>.<Schema>.<TableName>", Constants.INTERNAL)
```

To use a SQL pool external table, you need to have an EXTERNAL DATA SOURCE and an EXTERNAL FILE FORMAT that exists on the pool using the following examples:

```
--For an external table, you need to pre-create the data source and file
format in dedicated SQL pool using SQL queries:
CREATE EXTERNAL DATA SOURCE <DataSourceName>
WITH
    ( LOCATION = 'abfss://...' ,
      TYPE = HADOOP
    ) ;

CREATE EXTERNAL FILE FORMAT <FileFormatName>
WITH (
    FORMAT_TYPE = PARQUET,
    DATA_COMPRESSION = 'org.apache.hadoop.io.compress.SnappyCodec'
) ;
```

It is not necessary to create an EXTERNAL CREDENTIAL object if you are using Azure AD pass-through authentication from the storage account. The only thing you need to keep in mind is that you need to be a member of the "Storage Blob Data Contributor" role on the storage account. The next step is to use the df.write command within Scala with DATA_SOURCE, FILE_FORMAT, and the sqlAnalytics command in a similar way to writing data to an internal table.

The example is shown below:

```
df.write.
    option(Constants.DATA_SOURCE, <DataSourceName>).
    option(Constants.FILE_FORMAT, <FileFormatName>).
    sqlAnalytics("<DBName>.<Schema>.<TableName>", Constants.EXTERNAL)
```

Authenticate between spark and SQL pool in Azure Synapse Analytics

Another way to authenticate is using SQL Authentication, instead of Azure Active Directory (Azure AD) with the Azure Synapse Apache Spark Pool to Synapse SQL connector.

Currently, the Azure Synapse Apache Spark Pool to Synapse SQL connector does not support a token-based authentication to a dedicated SQL pool that is outside of the workspace of Synapse Analytics. In order to establish and transfer data to a dedicated SQL pool that is outside of the workspace without Azure AD, you would have to use SQL Authentication.

To read data from a dedicated SQL pool outside your workspace without Azure AD, you use the Read API. The Read API works for Internal tables (Managed Tables) and External Tables in the dedicated SQL pool.

The Read API looks as follows when using SQL Authentication:

```
val df = spark.read.  
option(Constants.SERVER, "samplews.database.windows.net") .  
option(Constants.USER, <SQLServer Login UserName>) .  
option(Constants.PASSWORD, <SQLServer Login Password>) .  
sqlanalytics("<DBName>.<Schema>.<TableName>")
```

The parameters it takes in are:

- **Constants.Server**: specify the URL of the server
- **Constants.USER**: SQLServer Login UserName
- **Constants.PASSWORD**: SQLServer Login Password
- **DBName**: the name of the database.
- **Schema**: the schema definition such as dbo.
- **TableName**: the name of the table you want to read data from.

In order to write data to a dedicated SQL Pool, you use the Write API. The Write API creates the table in the dedicated SQL pool, and then uses Polybase to load the data into the table that was created.

The Write API using SQL Auth looks as follows:

```
df.write.  
option(Constants.SERVER, "samplews.database.windows.net") .  
option(Constants.USER, <SQLServer Login UserName>) .  
option(Constants.PASSWORD, <SQLServer Login Password>) .  
sqlanalytics("<DBName>.<Schema>.<TableName>", <TableType>)
```

The parameters it takes in are:

- **Constants.Server**: specify the URL of the server
- **Constants.USER**: SQLServer Login UserName
- **Constants.PASSWORD**: SQLServer Login Password
- **DBName**: the name of the database.
- **Schema**: the schema definition such as dbo.
- **TableName**: the name of the table you want to read data from.
- **TableType**: specification of the type of table, which can have two values.
 - **Constants.INTERNAL** - Managed table in dedicated SQL pool
 - **Constants.EXTERNAL** - External table in dedicated SQL pool

Exercise: Integrate SQL and spark pools in Azure Synapse Analytics

In the following exercise, we will explore integrating SQL and Apache Spark pools in Azure Synapse Analytics.

Integrating SQL and Apache Spark pools in Azure Synapse Analytics

You want to write to a dedicated SQL pool after performing data engineering tasks in Spark, then reference the SQL pool data as a source for joining with Apache Spark DataFrames that contain data from other files.

You decide to use the Azure Synapse Apache Spark to Synapse SQL connector to efficiently transfer data between Spark pools and SQL pools in Azure Synapse.

Transferring data between Apache Spark pools and SQL pools can be done using JavaDataBaseConnectivity (JDBC). However, given two distributed systems such as Apache Spark and SQL pools, JDBC tends to be a bottleneck with serial data transfer.

The Azure Synapse Apache Spark pool to Synapse SQL connector is a data source implementation for Apache Spark. It uses the Azure Data Lake Storage Gen2 and PolyBase in SQL pools to efficiently transfer data between the Spark cluster and the Synapse SQL instance.

1. We have been using Python code in these cells up to this point. If we want to use the Apache Spark pool to Synapse SQL connector (`sqlanalytics`), one option is to create a temporary view of the data within the DataFrame. Execute the code below in a new cell to create a view named `top_purchases`:

```
# Create a temporary view for top purchases so we can load from Scala  
topPurchases.createOrReplaceTempView("top_purchases")
```

We created a new temporary view from the `topPurchases` dataframe that we created earlier and which contains the flattened JSON user purchases data.

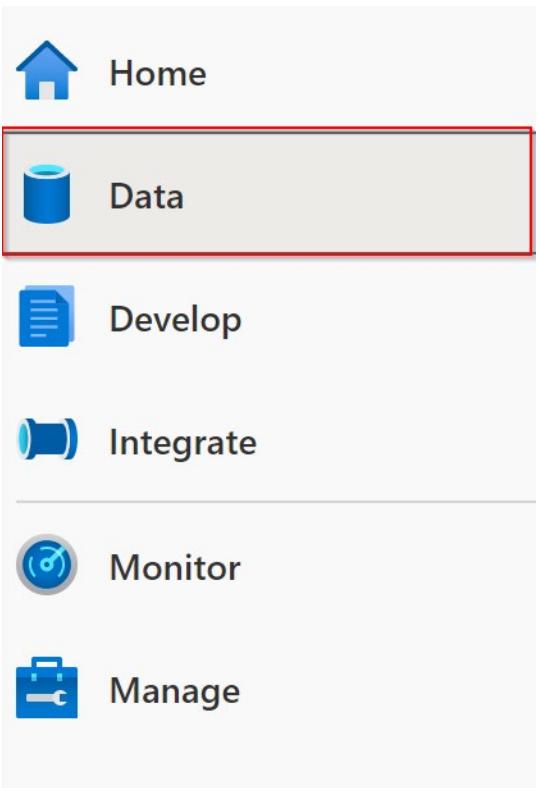
2. We must execute code that uses the Apache Spark pool to Synapse SQL connector in Scala. To do so, we add the `%%spark` magic to the cell. Execute the code below in a new cell to read from the `top_purchases` view:

```
%%spark  
// Make sure the name of the SQL pool (SQLPool01 below) matches the name of your SQL pool.  
val df = spark.sqlContext.sql("select * from top_purchases")  
df.write.sqlAnalytics("SQLPool01.wwi.TopPurchases", Constants.INTERNAL)
```

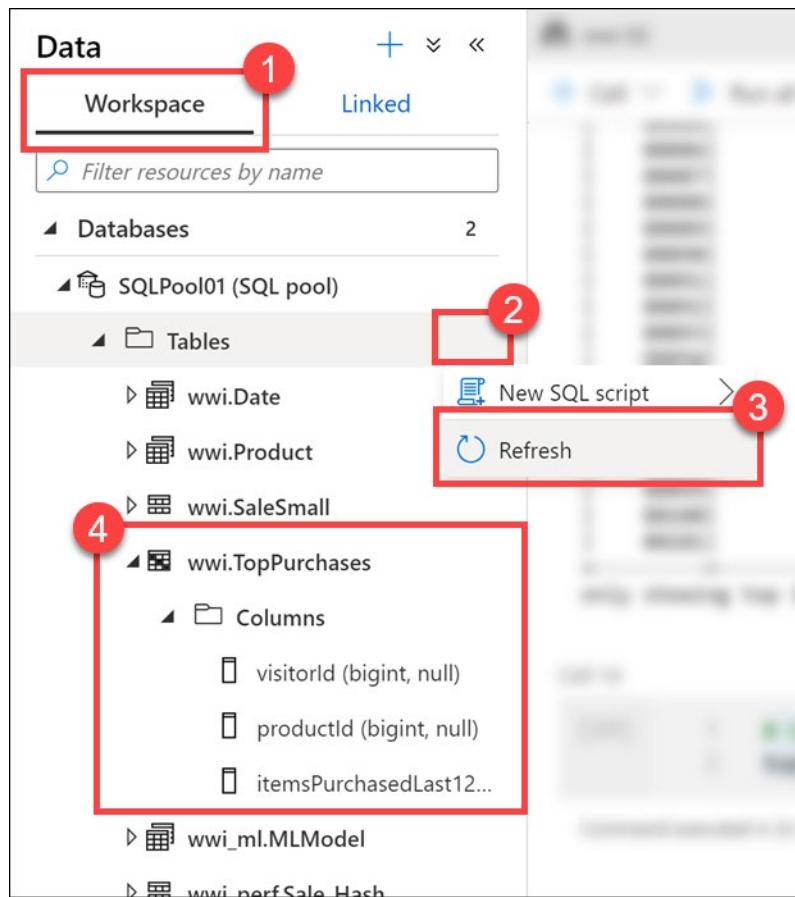
NOTE: The cell may take over a minute to execute. If you have run this command before, you will receive an error stating that "There is already an object named..." because the table already exists.

After the cell finishes executing, let's take a look at the list of SQL pool tables to verify that the table was successfully created for us.

3. **Leave the notebook open**, then navigate to the **Data** hub (if not already selected).



4. Select the **Workspace** tab (1), expand the SQL pool, select the **ellipses (...)** on Tables (2) and select **Refresh** (3). Expand the `wwi.TopPurchases` table and columns (4).



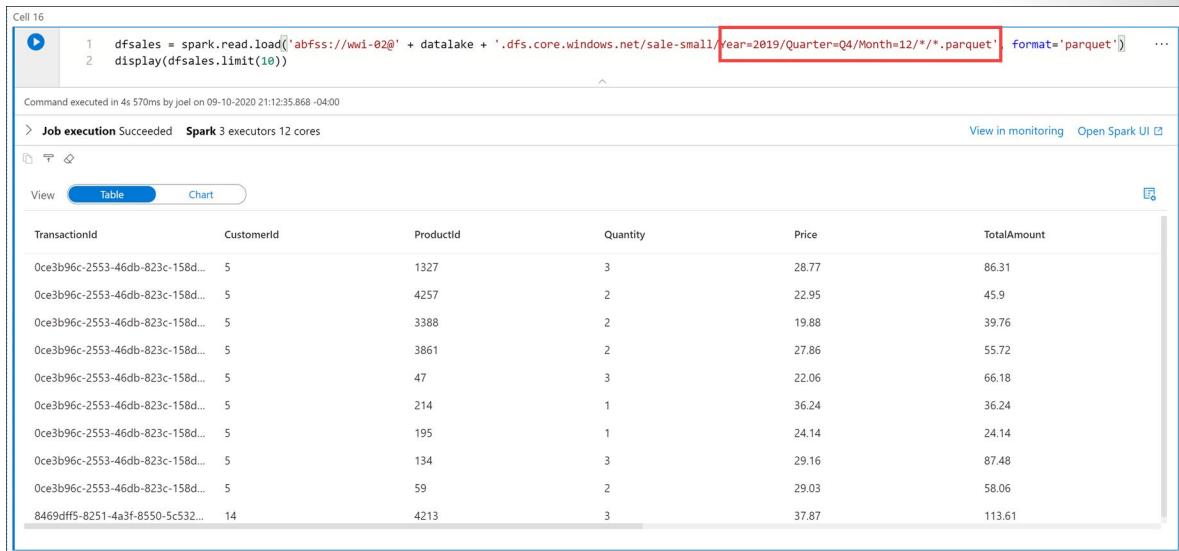
As you can see, the `wwi.TopPurchases` table was automatically created for us, based on the derived schema of the Apache Spark DataFrame. The Apache Spark pool to Synapse SQL connector was responsible for creating the table and efficiently loading the data into it.

5. **Return to the notebook** and execute the code below in a new cell to read sales data from all the Parquet files located in the `sale-small/Year=2019/Quarter=Q4/Month=12/` folder:

```
dfsales = spark.read.load('abfss://wwi-02@' + datalake + '.dfs.core.windows.net/sale-small/Year=2019/Quarter=Q4/Month=12/*/*.parquet', format='parquet')  
display(dfsales.limit(10))
```

NOTE: It can take over 3 minutes for this cell to execute.

The `datalake` variable we created in the first cell is used here as part of the file path.



Cell 16

```

1 dfsales = spark.read.load('abfss://wwi-02@' + datalake + '.dfs.core.windows.net/sale-small/year=2019/Quarter=Q4/Month=12/*.*.parquet') format='parquet' ...
2 display(dfsales.limit(10))

```

Command executed in 4s 570ms by joel on 09-10-2020 21:12:35.868 -04:00

> Job execution Succeeded Spark 3 executors 12 cores

View in monitoring Open Spark UI

Table

TransactionId	CustomerId	ProductId	Quantity	Price	TotalAmount
Oce3b96c-2553-46db-823c-158d...	5	1327	3	28.77	86.31
Oce3b96c-2553-46db-823c-158d...	5	4257	2	22.95	45.9
Oce3b96c-2553-46db-823c-158d...	5	3388	2	19.88	39.76
Oce3b96c-2553-46db-823c-158d...	5	3861	2	27.86	55.72
Oce3b96c-2553-46db-823c-158d...	5	47	3	22.06	66.18
Oce3b96c-2553-46db-823c-158d...	5	214	1	36.24	36.24
Oce3b96c-2553-46db-823c-158d...	5	195	1	24.14	24.14
Oce3b96c-2553-46db-823c-158d...	5	134	3	29.16	87.48
Oce3b96c-2553-46db-823c-158d...	5	59	2	29.03	58.06
8469dff5-8251-4a3f-8550-5c532...	14	4213	3	37.87	113.61

Compare the file path in the cell above to the file path in the first cell. Here we are using a relative path to load **all December 2019 sales** data from the Parquet files located in `sale-small`, vs. just December 31, 2010 sales data.

Next, let's load the `TopSales` data from the SQL pool table we created earlier into a new Apache Spark DataFrame, then join it with this new `dfsales` DataFrame. To do so, we must once again use the `%%spark` magic on a new cell since we'll use the Apache Spark pool to Synapse SQL connector to retrieve data from the SQL pool. Then we need to add the DataFrame contents to a new temporary view so we can access the data from Python.

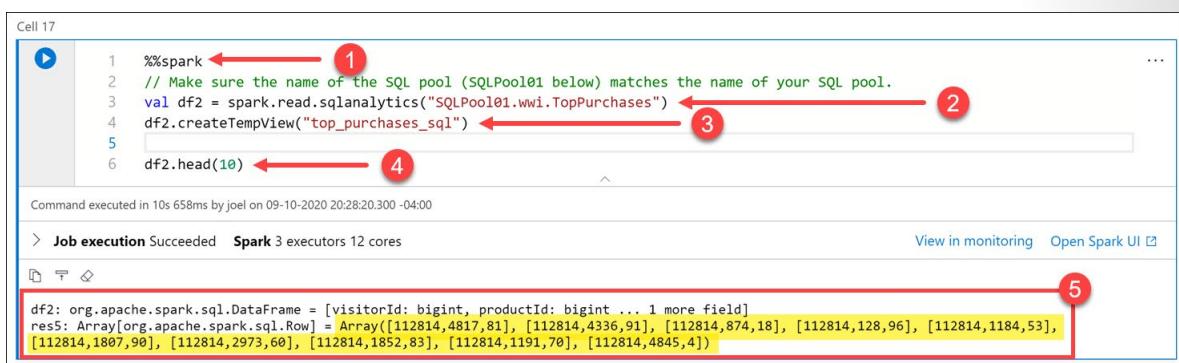
6. Execute the code below in a new cell to read from the `TopSales` SQL pool table and save it to a temporary view:

```

%%spark
// Make sure the name of the SQL pool (SQLPool01 below) matches the name of your SQL pool.
val df2 = spark.read.sqlanalytics("SQLPool01.wwi.TopPurchases")
df2.createTempView("top_purchases_sql")

df2.head(10)

```



Cell 17

```

1 %%spark
2 // Make sure the name of the SQL pool (SQLPool01 below) matches the name of your SQL pool.
3 val df2 = spark.read.sqlanalytics("SQLPool01.wwi.TopPurchases")
4 df2.createTempView("top_purchases_sql")
5 df2.head(10)

```

Command executed in 10s 658ms by joel on 09-10-2020 20:28:20.300 -04:00

> Job execution Succeeded Spark 3 executors 12 cores

View in monitoring Open Spark UI

```

df2: org.apache.spark.sql.DataFrame = [visitorId: bigint, productId: bigint ... 1 more field]
res5: Array[org.apache.spark.sql.Row] = Array([112814,4817,81], [112814,4336,91], [112814,874,18], [112814,128,96], [112814,1184,53],
[112814,1807,98], [112814,2973,60], [112814,1852,83], [112814,1191,70], [112814,4845,4])

```

The cell's language is set to Scala by using the `%%spark` magic (1) at the top of the cell. We declared a new variable named `df2` as a new DataFrame created by the `spark.read.sqlanalytics` method, which reads from the `TopPurchases` table (2) in the SQL pool. Then we populated a new

temporary view named `top_purchases_sql`(3). Finally, we showed the first 10 records with the `df2.head(10)` line (4). The cell output displays the DataFrame values (5).

7. Execute the code below in a new cell to create a new DataFrame in Python from the `top_purchases_sql` temporary view, then display the first 10 results:

```
dfTopPurchasesFromSql = sqlContext.table("top_purchases_sql")
```

```
display(dfTopPurchasesFromSql.limit(10))
```

The screenshot shows a Jupyter Notebook cell titled "Cell 18". The code in the cell is:

```
1 dfTopPurchasesFromSql = sqlContext.table("top_purchases_sql")
2
3 display(dfTopPurchasesFromSql.limit(10))
```

Below the code, it says "Command executed in 10s 320ms by joel on 09-10-2020 20:40:45.435 -04:00". The status bar indicates "**> Job execution Succeeded** Spark 3 executors 12 cores". There are links to "View in monitoring" and "Open Spark UI".

The output section shows a table with three columns: visitorId, productId, and itemsPurchasedLast12Months. The data is as follows:

visitorId	productId	itemsPurchasedLast12Months
118119	886	83
118119	2284	61
118119	1353	20
118119	4258	91
118119	2197	97
118119	3351	80
118119	3138	34
118119	2024	54
118120	2177	20
118120	831	28

8. Execute the code below in a new cell to join the data from the sales Parquet files and the TopPurchases SQL pool:

```
inner_join = dfsales.join(dfTopPurchasesFromSql,
    (dfsales.CustomerId == dfTopPurchasesFromSql.visitorId) & (dfsales.ProductId == dfTopPurchasesFromSql.productId))
```

```
inner_join_agg = (inner_join.select("CustomerId","TotalAmount","Quantity","itemsPurchasedLast12Months","top_purchases_sql.productId")
    .groupBy(["CustomerId","top_purchases_sql.productId"])
    .agg(
        sum("TotalAmount").alias("TotalAmountDecember"),
        sum("Quantity").alias("TotalQuantityDecember"),
        sum("itemsPurchasedLast12Months").alias("TotalItemsPurchasedLast12Months"))
    .orderBy("CustomerId"))
```

```
display(inner_join_agg.limit(100))
```

In the query, we joined the `dfsales` and `dfTopPurchasesFromSql` DataFrames, matching on `CustomerId` and `ProductId`. This join combined the `TopPurchases` SQL pool table data with the December 2019 sales Parquet data (1).

We grouped by the `CustomerId` and `ProductId` fields. Since the `ProductId` field name is ambiguous (it exists in both DataFrames), we had to fully qualify the `ProductId` name to refer to the one in the `TopPurchases` DataFrame (2).

Then we created an aggregate that summed the total amount spent on each product in December, the total number of product items in December, and the total product items purchased in the last 12 months (3).

Finally, we displayed the joined and aggregated data in a table view.

Feel free to select the column headers in the Table view to sort the result set.

```

Cell 20
1 inner_join = dfsales.join(dfTopPurchasesFromSql,
2   (dfsales.CustomerId == dfTopPurchasesFromSql.visitorId) & (dfsales.ProductId == dfTopPurchasesFromSql.productId))
3
4 inner_join_agg = (inner_join.select("CustomerId", "TotalAmount", "Quantity", "itemsPurchasedLast12Months", "top_purchases_sql.productId")
5   .groupBy(["CustomerId", "top_purchases_sql.productId"]))
6   .agg(
7     sum("TotalAmount").alias("TotalAmountDecember"),
8     sum("Quantity").alias("TotalQuantityDecember"),
9     sum("itemsPurchasedLast12Months").alias("TotalItemsPurchasedLast12Months"))
10   .orderBy("CustomerId") )
11
12 display(inner_join_agg.limit(100))
```

1
2
3
4

Command executed in 40s 941ms by joel on 09-10-2020 21:40:40.798 -0:00

> Job execution Succeeded Spark 3 executors 12 cores

View Table Chart

CustomerId	productId	TotalAmountDecember	TotalQuantityDecember	TotalItemsPurchasedLast12Months
80034	70	34.32	1	73
80097	1160	105.04	4	46
80126	30	62.4	3	95
80145	4475	21.66	1	40
80167	4097	69.99	3	23
80168	169	77.58	2	45

Externalize the use of spark pools within Azure Synapse Workspace

You can allow other users to use the Azure Synapse Apache Spark to Synapse SQL connector in your Azure Synapse workspace.

First of all, it is necessary to be a Storage Blob Data Owner in relation to the Azure Data Lake Gen 2 storage account that is connected to your workspace. The reason why the user account has to be a member of that role is so that you can alter missing permissions for others.

In addition to the above, the user needs to have access to the Azure Synapse workspace. Finally, in order to allow other users to use the connector, it's imperative that the user has permissions to run the notebooks.

Below you'll find the options for you to set the right permissions by configuring Access Control Lists (ACLs) on the folder structure as follows.

Folder	/	synapse	workspaces	<workspace-name>	sparkpools	<sparkpool-name>	sparkpool-instances
Access Permissions	-X	-X	-X	-X	-X	-X	-WX
Default Permissions	—	—	—	—	—	—	—

You can configure the ACLs for all folders from "synapse" and downward from Azure portal.

If you want to configure the ACLs from the root "/" folder, there are some extra instructions you should follow:

1. Connect to the storage account that is connected to the Azure Synapse workspace. You can use Azure Storage Explorer to do so.
2. Select your Account and give the Azure Data Lake Storage Gen 2 URL, and the default file system for the Azure Synapse workspace.
3. If you see the storage account listed, right-click on the listing workspace and make sure you select "Manage Access".
4. Add the user to the root "/" folder with "Execute" access permission and select "Ok".

Transfer data outside the synapse workspace using the PySpark connector

You can transfer data to and from a dedicated SQL pool using a Pyspark Connector, which currently works with Scala.

Let's say that you have created or loaded a DataFrame called "pyspark_df", and then assume that you want to write that DataFrame into the data warehouse. How would you go about that task?

The first thing to do is to create a temporary table in a DataFrame in PySpark using the `createOrReplaceTempView` method

```
pyspark_df.createOrReplaceTempView("pysparkdftemptable")
```

The parameter that is passed through is the temporary table name, which in this case is called: "pysparkdftemptable". We are still using the `pyspark_df` DataFrame as you can see in the beginning of the statement.

Next, you would have to run a Scala cell in the PySpark notebook using magics (since we're using different languages, and it will only work in Scala):

```
%%spark
val scala_df = spark.sqlContext.sql ("select * from pysparkdftemptable")
scala_df.write.sqlAnalytics("sqlpool dbo PySparkTable", Constants.INTERNAL)
```

By using "val scala_df", we create a fixed value for the `scala_dataframe`, and then use the statement "select * from pysparkdftemptable", that returns all the data that we created in the temporary table in the previous step, and storing it in a table named `sqlpool dbo PySparkTable`

In the second line of the code, we specified following parameters:

- **DBName:** The database name that in the example above is named sqlpool
- **Schema:** The schema name that in the example above is named dbo
- **TableName:** The table name that in the example above is named PySparkTable
- **TableType:** Specifies the type of table, which has the value Constants.INTERNAL, which related to a managed table in the dedicated SQL pool.

Should you wish to read data using the PySpark connector, keep in mind that you read the data using scala first, then write it into a temporary table. Finally you use the Spark SQL in PySpark to query the temporary table into a DataFrame.

Knowledge check

Question 1

In what language can the Azure Synapse Apache Spark to Synapse SQL connector can be used?

- Python.
- SQL.
- Scala.

Question 2

When is it unnecessary to use import statements for transferring data between a dedicated SQL and Apache Spark pool?

- Use the integrated notebook experience from Azure Synapse Studio.
- Use the PySpark connector.
- Use token-based authentication.

Summary

In this module, you have learned how to integrate SQL and Apache Spark pools in Azure Synapse Analytics.

Now that you have completed this module, you are able to:

- Describe the integration methods between SQL and Apache Spark pools in Azure Synapse Analytics
- Understand the use cases for SQL and Apache Spark pools integration
- Authenticate in Azure Synapse Analytics
- Transfer data between SQL and Apache Spark pools in Azure Synapse Analytics
- Authenticate between Apache Spark and SQL pools in Azure Synapse Analytics
- Integrate SQL and Apache Spark pools in Azure Synapse Analytics
- Externalize the use of Apache Spark pools within Azure Synapse workspace
- Transfer data outside the synapse workspace using the PySpark connector

Monitor manage data engineering workloads with Apache Spark Azure Synapse Analytics

Introduction

In this module, you will learn how to monitor and manage data engineering workloads with Apache Spark in Azure Synapse Analytics.

After completing this module, you will be able to:

- Monitor Apache Spark pools in Azure Synapse Analytics
- Understand resource utilization of Spark pools in Azure Synapse Analytics
- Monitor query activity of Apache Spark pools in Azure Synapse Analytics
- Base-line Apache Spark performance with Apache Spark history server in Azure Synapse Analytics
- Optimize Apache Spark jobs in Azure Synapse Analytics
- Automate scaling of Apache Spark pools in Azure Synapse Analytics

Prerequisites

Before taking this module, it is recommended that you complete the following modules:

- Azure Data Fundamentals
- Introduction to Azure Data Factory
- Introduction to Azure Synapse Analytics

Monitor Spark Pools Analytics

If you want to monitor your Apache Spark pool, the best place to go is the **Monitor** tab in Azure Synapse Studio.

The Monitor hub enables you to view various Azure Synapse Analytics activities including pipeline and trigger runs, and the status of the various integration runtimes that are running. You can also view Apache Spark jobs, SQL requests, and data flow debug activities.

The Monitor hub is your first stop for debugging issues and gaining insights on the resource usage of your Apache Spark pool. You can see the history of all the activities that have taken place in the workspace, and which ones are currently active.

You may have Apache Spark pool activities that are part of Azure Synapse Pipeline activities. If you have created a pipeline, and you ran that pipeline, you can see all the pipeline run activities here. It is also possible to view run details where you can see inputs and outputs for the activities in the pipeline and any error messages. If you automated a pipeline run by setting up automated triggers, you can find the runs here as well.

In relation to Apache Spark applications, you can see all the Apache Spark applications that are running or have run in your workspace. Let's say you ran some Apache Spark activities, what do you do for monitoring?

First, within Azure Synapse Studio, you should navigate to the **Monitor** hub, the select **Activities**, and select **Apache Spark applications**. It's here where you can see all the Apache Spark applications that are running or have run in your workspace.

If you want to find out more information about an Apache Spark Application that is no longer running, you should select the application name in the **Monitor** hub, select the **Apache Spark Application** name. Here you will find all the details of the Apache Spark application.

If you are familiar with Apache Spark, you can find the standard Apache Spark history server UI by clicking on Apache Spark history server. Not only can you check the diagnostics of the Apache Spark application that ran, such as a notebook attached to an Apache Spark pool, but you can also check the logs if you navigate to the logs tab:

On the right-hand side, you'll find the details of the Apache Spark pool application and the running duration, number of executors, the Apache Spark pool details, and much more. Additionally, if you want to view details for each stage, you can go to the View details tab of one of the stages that looks like the follows:

The screenshot shows the Apache Spark UI interface. At the top, there's a navigation bar with tabs: Jobs, Stages (which is selected), Storage, Environment, Executors, Graph (Preview), Diagnostic (Preview), and SQL. Below the navigation bar, the page title is "Details for Stage 0 (Attempt 0)". Under this, there are sections for "Total Time Across All Tasks: 2 s" and "Locality Level Summary: Process local: 1". There are three expandable sections: "DAG Visualization", "Show Additional Metrics", and "Event Timeline". A table titled "Summary Metrics for 1 Completed Tasks" follows, showing metrics like Duration (2 s) and GC Time (69 ms). Below this is another expandable section "Aggregated Metrics by Executor", which contains a table with columns: Executor ID, Address, Task Time, Total Tasks, Failed Tasks, Killed Tasks, Succeeded Tasks, and Blacklisted. The table shows one executor (ID 2) with address 14f366e5779a4536ad399f79da5b562a03294a19985, task time 3 s, total tasks 1, failed tasks 0, killed tasks 0, succeeded tasks 1, and blacklisted status false. Finally, there's a section "Tasks (1)" with a table showing details for a single task (Index 0) with ID 0, Attempt 0, Status SUCCESS, Locality Level PROCESS_LOCAL, Executor ID 2, Host 14f366e5779a4536ad399f79da5b562a03294a19985, Launch Time 2020/11/17 07:28:15, Duration 2 s, GC Time 69 ms, and Errors 0.

It will redirect you to the Apache Spark UI where you can find more details in relation to the stages of the Apache Spark Pool. You can also check details of all the stages by selecting **Spark UI**.

Base line Apache Spark performance history server Azure Synapse Analytics

The Apache Spark history server can be used to debug and diagnose completed and running Apache Spark applications. You can use the Apache Spark history server web UI from the Azure Synapse Studio environment.

Once you launch it, there are several tabs that you can use in order to monitor the Apache Spark application:

- Jobs
- Stages
- Storage
- Environment
- Executors
- SQL

The Apache Spark history server is the web user interface known as the Spark UI, and is used to view completed and running Apache Spark applications. If you want to navigate to the Apache Spark History server, you can navigate to the Azure Synapse Analytics Studio environment and go to the Monitor tab. In the Monitor tab, you can select 'Apache Spark Applications'.

The screenshot shows the Azure Synapse Studio interface under the 'Integration' tab. In the center, it displays the 'Apache Spark applications' section for a specific job. The job details are as follows:

- Livy ID:** Synapse_NBS_sparkpoolmod_1605597928
- Completed tasks:** 16 of 16
- Status:** Cancelled
- Total duration:** 33m 31s

The job consists of three stages:

- Stage 0:** 1 task, Duration: 3s 44ms, Rows: 0, Data read: 0B/s, Data written: 0B/s.
- Stage 1:** 4 tasks, Duration: 3s 720ms, Rows: 0, Data read: 0B/s, Data written: 0B/s.
- Stage 2:** 11 tasks, Duration: 121ms, Rows: 0, Data read: 0B/s, Data written: 0B/s.

The 'Diagnostics' tab is selected, showing the following results:

- Failed jobs:** All 3 jobs were completed successfully.
- Data skew:** No data skew detected.
- Time skew:** No time skew detected.
- Executor utilization:** (No details shown)

The 'Logs' tab is also present but not selected.

If you are familiar with Apache Spark, you can find the standard Apache Spark history server UI by selecting Open Spark UI.

Another way to open the Apache Spark History server is to navigate to the Data tab, where if you create a notebook and read a DataFrame you can go to the bottom of the page and find the Spark History Server known as the Spark UI.

- From your Azure Synapse Studio notebook, select **Open Spark UI** from the job execution output cell or from the status panel at the bottom of the notebook document.

The screenshot shows the Azure Synapse Studio Data tab with a notebook titled 'Notebook 1'. The notebook content is as follows:

```

1 new_rows = [('CA',22,45000),('WA',35,65000),('WA',50,85000)]
2 demo_df = spark.createDataFrame(new_rows,['state','age','salary'])
3 demo_df.show()

```

The notebook output shows the command was executed in 10s 187ms by kawajie on 11-20-2020 12:27:40.494 +0000.

The status panel at the bottom of the notebook shows the following for three jobs:

ID	Description	Status	Stages	Tasks	Submission Time	Duration
> Job 0	showString at NativeMethodAccessorImpl.java:0	Succeeded	1/1	1/1 succeeded	12:27:32 PM, 11/20/20	3 sec
> Job 1	showString at NativeMethodAccessorImpl.java:0	Succeeded	1/1	4/4 succeeded	12:27:36 PM, 11/20/20	3 sec
> Job 2	showString at NativeMethodAccessorImpl.java:0	Succeeded	1/1	11/11 succeeded	12:27:39 PM, 11/20/20	0 sec

A red box highlights the 'Open Spark UI' link in the 'Status' column of the last row.

- Select **Spark UI** from the slide out panel and you'll be redirected to the Spark monitoring tab where you will land in the Jobs tab.

Spark Jobs (?)

User: trusted-service-user
Total Uptime: 4.7 min
Scheduling Mode: FIFO
Completed Jobs: 3

► Event Timeline
▼ Completed Jobs (3)

Job Id (Job Group)	Description	Submitted	Duration	Stages: Succeeded/Total	Tasks (for all stages): Succeeded/Total
2 (3)	Job group for statement 3 <code>showString at NativeMethodAccessorImpl.java:0</code>	2020/11/23 07:11:20	0.1 s	1/1	11/11
1 (3)	Job group for statement 3 <code>showString at NativeMethodAccessorImpl.java:0</code>	2020/11/23 07:11:17	3 s	1/1	4/4
0 (3)	Job group for statement 3 <code>showString at NativeMethodAccessorImpl.java:0</code>	2020/11/23 07:11:13	4 s	1/1	1/1

- Within the Jobs tab of the Spark History server, you can see the Job ID, Description, the time when the job was submitted, the duration, the stages, and the task (for all stages). If you select a job ID and go to the description and select the url, you'll be redirected to this screen:

Details for Job 2

Status: SUCCEEDED
Job Group: 3
Completed Stages: 1

► Event Timeline
▼ DAG Visualization

Stage 2

```

graph TD
    A[parallelize] --> B[mapPartitions]
    B --> C[map]
    C --> D[mapPartitions]
    D --> E[Scan ExistingRDD]
    E --> F[WholeStageCodegen]
  
```

It will give you a DAG Visualization of the stage.

- If you select the Stages tab, you'll find all the completed stages. If you want to dig deeper into the stages, you can select the description url as shown below:

The screenshot shows the Apache Spark UI interface with the 'Stages' tab selected. The title bar indicates the version is 2.4.4.2.6.98.201-25973884 and the application ID is Synapse_sparkpoolmod_1606115314. The 'Completed Stages' section displays three completed stages:

Stage Id	Description	Submitted	Duration	Tasks: Succeeded/Total	Input	Output	Shuffle Read	Shuffle Write
2	Job group for statement 3 showString at NativeMethodAccessorImpl.java:0	2020/11/23 07:11:20	0.1 s	11/11				
1	Job group for statement 3 showString at NativeMethodAccessorImpl.java:0	2020/11/23 07:11:17	3 s	4/4				
0	Job group for statement 3 showString at NativeMethodAccessorImpl.java:0	2020/11/23 07:11:13	4 s	1/1				

- In the Storage tab, your Resilient Distributed Datasets (RDDs) will only be cached once you have evaluated it. A common way of forcing evaluation and populate a cache is, for example, to call the 'count' command.

`demo_df.cache()`

`demo_df.count()`

- The image shows what you would see in the Storage tab of the Apache Spark UI:

The screenshot shows the Apache Spark UI interface with the 'Storage' tab selected. The title bar indicates the version is 2.4.4.2.6.98.201-25973884 and the application ID is Synapse_sparkpoolmod_1606115314. The 'RDDs' section displays one RDD:

ID	RDD Name	Storage Level	Cached Partitions	Fraction Cached	Size in Memory	Size on Disk
22	Scan ExistingRDD[state#0.age#1L,salary#2L]	Memory Deserialized 1x Replicated	16	100%	2.4 KB	0.0 B

- If you want to see more details, you can go into the RDD Name url where you'll be shown these details:

RDD Storage Info for Scan ExistingRDD[state#0,age#1L,salary#2L]

Storage Level: Memory Deserialized 1x Replicated
Cached Partitions: 16
Total Partitions: 16
Memory Size: 2.4 KB
Disk Size: 0.0 B

Data Distribution on 2 Executors

Host	On Heap Memory Usage	Off Heap Memory Usage	Disk Usage
bf5e6d37de2247e7847bed207a7b5e8f00337162244:36557	888.0 B (29.7 GB Remaining)	0.0 B (0.0 B Remaining)	0.0 B
bf5e6d37de2247e7847bed207a7b5e8f004f3689053:41025	1584.0 B (29.7 GB Remaining)	0.0 B (0.0 B Remaining)	0.0 B

16 Partitions

Block Name	Storage Level	Size in Memory	Size on Disk	Executors
rdd_22_0	Memory Deserialized 1x Replicated	24.0 B	0.0 B	bf5e6d37de2247e7847bed207a7b5e8f00337162244:36557
rdd_22_1	Memory Deserialized 1x Replicated	24.0 B	0.0 B	bf5e6d37de2247e7847bed207a7b5e8f004f3689053:41025
rdd_22_10	Memory Deserialized 1x Replicated	720.0 B	0.0 B	bf5e6d37de2247e7847bed207a7b5e8f00337162244:36557
rdd_22_11	Memory Deserialized 1x Replicated	24.0 B	0.0 B	bf5e6d37de2247e7847bed207a7b5e8f004f3689053:41025
rdd_22_12	Memory Deserialized 1x Replicated	24.0 B	0.0 B	bf5e6d37de2247e7847bed207a7b5e8f00337162244:36557
rdd_22_13	Memory Deserialized 1x Replicated	24.0 B	0.0 B	bf5e6d37de2247e7847bed207a7b5e8f004f3689053:41025
rdd_22_14	Memory Deserialized 1x Replicated	24.0 B	0.0 B	bf5e6d37de2247e7847bed207a7b5e8f00337162244:36557
rdd_22_15	Memory Deserialized 1x Replicated	720.0 B	0.0 B	bf5e6d37de2247e7847bed207a7b5e8f004f3689053:41025
rdd_22_2	Memory Deserialized 1x Replicated	24.0 B	0.0 B	bf5e6d37de2247e7847bed207a7b5e8f00337162244:36557
rdd_22_3	Memory Deserialized 1x Replicated	24.0 B	0.0 B	bf5e6d37de2247e7847bed207a7b5e8f004f3689053:41025
rdd_22_4	Memory Deserialized 1x Replicated	24.0 B	0.0 B	bf5e6d37de2247e7847bed207a7b5e8f00337162244:36557
rdd_22_5	Memory Deserialized 1x Replicated	720.0 B	0.0 B	bf5e6d37de2247e7847bed207a7b5e8f004f3689053:41025

- If we move on to the Environment tab, you'll get the Runtime Information and the Spark Properties, System properties, and the class path entries.
- The next thing we will look at is the Executors tab. Here you can find all the information about the executors:

Executors

Summary

RDD Blocks	Storage Memory	Disk Used	Cores	Active Tasks	Failed Tasks	Complete Tasks	Total Tasks	Task Time (GC Time)	Input	Shuffle Read	Shuffle Write	Blacklisted
Active(3)	97.7 KB / 95.6 GB	0.0 B	16	0	0	67	67	22 s (0.2 s)	0.0 B	2.7 KB	2.7 KB	0
Dead(0)	0.0 B / 0.0 B	0.0 B	0	0	0	0	0	0 ms (0 ms)	0.0 B	0.0 B	0.0 B	0
Total(3)	97.7 KB / 95.6 GB	0.0 B	16	0	0	67	67	22 s (0.2 s)	0.0 B	2.7 KB	2.7 KB	0

Executors

Executor ID	Address	Status	RDD Blocks	Storage Memory	Disk Used	Cores	Active Tasks	Failed Tasks	Complete Tasks	Total Tasks	Task Time (GC Time)	Input	Shuffle Read	Shuffle Write	Logs	Thread Dump
driver	bf5e6d37de2247e7847bed207a7b5e8f000a8796545:38719	Active	0	35.7 KB / 31.9 GB	0.0 B	0	0	0	0	0	0 ms (0 ms)	0.0 B	0.0 B	0.0 B	stdout stderr	Thread Dump
1	bf5e6d37de2247e7847bed207a7b5e8f004f3689053:41025	Active	8	29.3 KB / 31.9 GB	0.0 B	8	0	0	33	33	12 s (0.1 s)	0.0 B	905 B	1.4 KB	Thread Dump	Thread Dump
2	bf5e6d37de2247e7847bed207a7b5e8f00337162244:36557	Active	8	32.6 KB / 31.9 GB	0.0 B	8	0	0	34	34	10 s (61 ms)	0.0 B	1.8 KB	1.4 KB	Thread Dump	Thread Dump

Showing 1 to 3 of 3 entries

Previous 1 Next

- Finally the SQL tab. Here you can find the completed queries and details within the IDs for the queries.

The screenshot shows the Apache Spark UI interface with the 'SQL' tab selected. At the top, there are tabs for Jobs, Stages, Storage, Environment, Executors, and SQL. Below the tabs, it says 'Completed Queries: 8'. A red box highlights the 'Completed Queries (8)' link. The main area displays a table of completed queries:

ID	Description	Submitted	Duration	Job IDs
-4	PeregrinePlanLogEvent Spark SQL Plans with annotations.	+details 2020/11/23 07:31:45	21 ms	
3	count at NativeMethodAccessorImpl.java:0 <pre>org.apache.spark.sql.Dataset.count(Dataset.scala:2843) sun.reflect.NativeMethodAccessorImpl.invoke(Native Method) sun.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAccessorImpl.java:43) java.lang.reflect.Method.invoke(Method.java:498) py4j.reflection.MethodInvoker.invoke(MethodInvoker.java:244) py4j.reflection.ReflectionEngine.invoke(ReflectionEngine.java:357) py4j.Gateway.invoke(Gateway.java:282) py4j.commands.AbstractCommand.invokeMethod(AbstractCommand.java:132) py4j.commands.CallCommand.execute(CallCommand.java:79) py4j.GatewayConnection.run(GatewayConnection.java:238) java.lang.Thread.run(Thread.java:748)</pre>	+details 2020/11/23 07:31:44	0.6 s	[5]
-3	PeregrinePlanLogEvent	+details 2020/11/23 07:30:58	12 ms	
2	count at NativeMethodAccessorImpl.java:0	+details 2020/11/23 07:30:58	0.3 s	[4]
-2	PeregrinePlanLogEvent	+details 2020/11/23 07:28:43	26 ms	
1	count at NativeMethodAccessorImpl.java:0	+details 2020/11/23 07:28:42	0.6 s	[3]
-1	PeregrinePlanLogEvent	+details 2020/11/23 07:11:20	0.1 s	
0	showString at NativeMethodAccessorImpl.java:0	+details 2020/11/23 07:11:12	7 s	[0][1][2]

Now we have been through all the different tabs, using the Spark UI, enabling you to optimize and baseline through the Apache Spark performance and settings.

Optimize Apache Spark jobs Analytics

Once you have checked the Monitor tab within the Azure Synapse Studio environment, and decide that you should improve the performance of an Apache Spark pool run, you have several areas to consider, including:

- Choosing the data abstraction
- Use the optimal data format
- Use the cache option
- Check the memory efficiency
- Use Bucketing
- Optimize Joins and Shuffles if appropriate
- Optimize Job Execution

To optimize the Apache Spark Jobs in Azure Synapse Analytics, you need to consider the cluster configuration for the workload you're running on that cluster. You might run into challenges such as memory pressure (if not configured appropriately by choosing the wrong size of executors), long running operations, and tasks that might result in cartesian operations.

If you want to speed up the jobs, you'd have to configure the appropriate caching for that task, and check joins and shuffles in relation to data skew. Therefore, it is imperative that you monitor and review Apache Spark Job executions that are long running or resource consuming. Some recommendations for you to optimize the Apache Spark Job include:

Choosing the data abstraction

Some of the earlier Apache Spark versions use Resilient Distributed DataSets (RDDs) to abstract the data. Apache Spark 1.3 and 1.6 introduced the use of DataFrames and DataSets. The following relative merits might help you to optimize in relation to your data abstraction:

DataFrames

Using DataFrames would be a great place to start. DataFrames provide query optimization through Catalyst. It also includes a whole-stage code generation with direct memory access. When you want to have the best developer-friendly experience, it might be better to use DataSets, since there are no compile-time checks or domain object programming.

DataSets are good in complex ETL pipeline optimization where the performance effect is acceptable. Just be cautious when using DataSets in aggregations, since it might affect the performance. It will provide query optimization through Catalyst and is developer-friendly by providing object programming and compile-time checks.

RDDs

It is not necessary to use RDDs unless you want or need to build a new custom RDD. However, there is no query optimization through Catalyst and no whole-stage code generation and would still have a high garbage collection (GC) overhead. The only way to use RDDs is with Apache Spark 1.x legacy APIs.

Use the optimal data format

Apache Spark supports many data formats. The formats that you can use are csv, json, xml, parquet etc. It can also be extended by other formats with external data sources. A useful tip is to use Parquet with snappy compression (which also happen to be the default in Apache Spark 2.x.) as it stores data in a columnar format, is compressed and highly optimized in Apache Spark, and is splittable to decompress.

Use the cache option

When it comes to the caching, there is a native built-in Apache Spark caching mechanism. It can be used through different methods like: `.persist()`, `.cache()`, and `CACHE TABLE`. When using small datasets, it might be effective.

In ETL pipelines where caching of intermediate results is necessary it might come in handy too. Just keep in mind that when you need to do partitioning, the Apache Spark native caching mechanism might have some downsides because a cached table won't keep the partitioning data.

Check the memory efficiency

It is also imperative to understand how to use the memory efficiently. Apache Spark operates by placing data in memory. Therefore, managing memory resources is an aspect of optimizing Apache Spark jobs executions.

One way to manage memory resources might be to check smaller data partitions and check data size, types, and distributions when you formulate a partitioning strategy. Another way to optimize is to

consider Kryo data serialization: **Kryo data serialization⁵**, versus the default Java serialization. Always bear in mind to keep monitoring and tuning the Apache Spark configuration settings.

Use Bucketing

Bucketing is almost the same as data partitioning. The way it differs is that a bucket holds a *set* of column values instead of one. It might work well when you partition on large (millions or more) values like product identifiers. A bucket is determined by hashing the bucket key of a row. Bucketed tables are optimized because it is a metadata operation on how the data is bucketed and sorted.

Some advanced bucketing features are:

- Query optimization based on bucketing meta-information.
- Optimized aggregations.
- Optimized joins.

However, bucketing doesn't exclude partitioning. They go hand in hand, and you can use partitioning and bucketing at the same time.

Optimize joins and shuffles

When you have a slower performance on join or shuffle jobs, it can be caused by data skew. Data skew is data that is stored asymmetrically on your system. An example might be that a job typically only takes 20 seconds, however running the same job where data is joined and shuffled can take hours.

To fix that data skew, you can salt the entire key, or use an *isolated salt* for only some subset of keys. Another option to investigate might be the introduction of a bucket column or pre-aggregated data in buckets.

However, there's more to causing slow performance just by joins alone, since it might be the join type that is causing the slow performance.

Apache Spark uses the `SortMerge` join type. This type of join is best suited for large data sets but is otherwise computationally expensive because it must first sort the left and right sides of data before merging them. Therefore, a `Broadcast` join might be better suited for smaller data sets, or where one side of the join is much smaller than the other side.

You can change the join type in your configuration by setting `spark.sql.autoBroadcastJoinThreshold`, or you can set a join hint using the DataFrame APIs (`dataframe.join(broadcast(df2))`) as shown in the following code.

```
// Option 1
spark.conf.set("spark.sql.autoBroadcastJoinThreshold", 1*1024*1024*1024)

// Option 2
val df1 = spark.table("FactTableA")
val df2 = spark.table("dimMP")
df1.join(broadcast(df2), Seq("PK")) .
    createOrReplaceTempView("V_JOIN")

sql("SELECT col1, col2 FROM V_JOIN")
```

⁵ <https://github.com/EsotericSoftware/kryo>

If you did decide to use bucketed tables, you will have a third join type, the `Merge` join. A correctly pre-partitioned and pre-sorted dataset will skip the expensive sort phase from a `SortMerge` join. Another thing to keep in mind is that the order of the different types of joins does matter, especially in complex queries. Therefore, it's advised to start with the most selective joins. In addition, try to move joins that increase the number of rows after aggregations when possible.

Optimize Job Execution

Looking at the sizing of executors to increase performance in your Apache Spark job, you could consider the Java garbage collection (GC) overhead and use the following factors to reduce executor size:

- Reduce heap size below 32 GB to keep GC overhead < 10%.
- Reduce the number of cores to keep GC overhead < 10%.

Or consider the following factors to increase executor size:

- Reduce communication overhead between executors.
- Reduce the number of open connections between executors (N^2) on larger clusters (> 100 executors).
- Increase heap size to accommodate for memory-intensive tasks.
- Reduce per-executor memory overhead.
- Increase utilization and concurrency by oversubscribing CPU.

As a rule of thumb, when selecting the executor size:

- Start with 30 GB per executor and distribute available machine cores.
- Increase the number of executor cores for larger clusters (> 100 executors).
- Modify size based both on trial runs and on the preceding factors such as GC overhead.

When running concurrent queries, consider as follows:

- Start with 30 GB per executor and all machine cores.
- Create multiple parallel Apache Spark applications by oversubscribing CPU (around 30% latency improvement).
- Distribute queries across parallel applications.
- Modify size based both on trial runs and on the preceding factors such as GC overhead.

As stated before, it's important to keep monitoring the performance, especially outliers, using the timeline view, SQL graph, job statistics, and so on. It might be a case where one of the executors is slower than the other, which most frequently happens on large clusters (30+ nodes). What you then might consider is to divide the work into more tasks so that the scheduler can compensate for the slower tasks.

If there is an optimization necessary in relation to the optimization of a job execution, make sure you keep in mind the caching (an example might be using the data twice, but cache it). If you broadcast variables on all the executors you set up, due to the variables only being serialized once, you'll have faster lookups.

In another case you might use the thread pool that runs on the driver, which could result in faster operations for many tasks.

Automate scaling of Apache Spark Pools Analytics

Within an Apache Spark Pool, it is possible to configure a fixed size when you disable autoscaling. When you enable autoscale, you can set a minimum and maximum number of nodes in order to control the scale that you'd like.

Once you have enabled autoscale, Azure Synapse Analytics will monitor the resources of the load, and it will scale the number of nodes up or down. There will be continuous monitoring depending on CPU usage, pending memory, free CPU, free memory, and the used memory per node when it comes to the metrics involved to make a decision to scale up or down. It checks these metrics every 30 seconds and makes scaling decisions based on the values. There's no extra charge for this feature.

Below is a table that shows the metrics that autoscale enablement of the Apache Spark Pool within Azure Synapse analytics instance checks and collects:

Metric	Description
Total Pending CPU	The total number of cores required to start execution of all pending nodes.
Total Pending Memory	The total memory (in MB) required to start execution of all pending nodes.
Total Free CPU	The sum of all unused cores on the active nodes.
Total Free Memory	The sum of unused memory (in MB) on the active nodes.
Used Memory per Node	The load on a node. A node on which 10 GB of memory is used is considered under more load than a worker with 2 GB of used memory.

When we look at load-based scale conditions, the autoscale functionality will issue a scale request based on the metrics outlined in the table below:

Scale-up	Scale-down
Total pending CPU is greater than total free CPU for more than 1 minute.	Total pending CPU is less than total free CPU for more than 2 minutes.
Total pending memory is greater than total free memory for more than 1 minute.	Total pending memory is less than total free memory for more than 2 minutes.

When autoscale scales up, it will calculate the number of new nodes that would be needed in order to meet the CPU and memory requirements. Next, it will issue the scale-up requests and add the number of nodes required to do the job.

In case autoscale performs the action of scaling down, the decision is based on the number of executors and the application primaries per node, and the CPU and memory requirements.

The autoscale functionality will then issue the request to remove some nodes. The autoscale functionality will also check which nodes are candidates for removal based on the current job execution. The scale down operation first decommissions the nodes, and then removes them from the cluster.

If you'd like to get started with the autoscale functionality, you'd have to follow the next steps:

Create a serverless Apache Spark pool with Autoscaling

To enable the Autoscale feature, complete the following steps as part of the normal Apache Spark pool creation process:

1. On the **Basics** tab, select the **Enable autoscale** checkbox.

2. Enter the desired values for the following properties:

- **Min** number of nodes.
- **Max** number of nodes.

The initial number of nodes will be the minimum. This value defines the initial size of the instance when it's created. The minimum number of nodes can't be fewer than three.

When considering the best practices to use for the autoscale feature, consider latency as part of the scale up or down operations. It could take 1 to 5 minutes in order for the scaling operations (whether that's scaling up or down) to complete. Also, when you scale down, the nodes will first be put in a decommissioned state such that there won't be new executors launching on the node. The jobs that are still running will continue to run and finish, but the pending jobs will be in a waiting state to be scheduled as normal but with fewer nodes.

Knowledge check

Question 1

What is one of the possible ways to optimize an Apache Spark Job?

- Remove all nodes.
- Remove the Apache Spark Pool.
- Use bucketing.

Question 2

What can cause a slower performance on join or shuffle jobs?

- Data skew.
- Enablement of autoscaling
- Bucketing.

Summary

In this module, you have learned how to monitor and manage data engineering workloads with Apache Spark in Azure Synapse Analytics.

Since you have completed this module, you are now able to:

- Monitor Apache Spark pools in Azure Synapse Analytics
- Understand resource utilization of Spark pools in Azure Synapse Analytics
- Monitor query activity of Spark pools in Azure Synapse Analytics
- Base-line Apache Spark performance with Apache Spark history server in Azure Synapse Analytics
- Optimize Apache Spark jobs in Azure Synapse Analytics
- Automate scaling of Apache Spark pools in Azure Synapse Analytics

Module Lab information

Lab 4 - Explore, transform, and load data into the Data Warehouse using Apache Spark

- **Estimated Time:** 50 - 60 minutes
- **Lab files:** The files for this lab are located in the *Instructions\Labs\05* folder.

Lab overview

This lab teaches the student how to explore data stored in a data lake, transform the data, and load data into a relational data store. The student will explore Parquet and JSON files and use techniques to query and transform JSON files with hierarchical structures. Then the student will use Apache Spark to load data into the data warehouse and join Parquet data in the data lake with data in the dedicated SQL pool.

Lab objectives

After completing this lab, you will be able to:

- Perform Data Exploration in Synapse Studio
- Ingest data with Spark notebooks in Azure Synapse Analytics
- Transform data with DataFrames in Spark pools in Azure Synapse Analytics
- Integrate SQL and Spark pools in Azure Synapse Analytics

Module summary

module-summary

In this module, you have learned how to perform data engineering tasks with Azure Synapse Apache Spark pools, which enables you to boost the performance of big-data analytic applications by in-memory cluster computing. You have explored the ingestion of data with Apache Spark notebooks, transformed data with DataFrames, integrated SQL and Apache Spark pools in Azure Synapse Analytics, whilst monitoring and managing the workloads.

Learning objectives

In this module, you have:

- Understood big data engineering workloads with Apache Spark pools in Azure Synapse Analytics
- Ingested data with Apache Spark notebooks in Azure Synapse Analytics
- Transformed data with DataFrames in Apache Spark pools in Azure Synapse Analytics
- Integrated SQL and Apache Spark pools in Azure Synapse Analytics
- Monitored and managed data engineering workloads with Apache Spark pools in Azure Synapse Analytics

Post Course Review

After the course, consider visiting **Azure Apache Spark for Azure Synapse Analytics⁶**. The Apache Spark in Azure Synapse Analytics provides an overview of how Apache Spark is integrated with Azure Synapse Analytics.

Important

Remember

Should you be working in your own subscription while running through this content, it is best practice at the end of a project to identify whether or not you still need the resources you created. Resources left running can cost you money.

You can delete resources one by one, or just delete the resource group to get rid of the entire set.

⁶ <https://docs.microsoft.com/en-us/azure/synapse-analytics/spark/apache-spark-overview>

Answers

Question 1

What is an element of an Apache Spark Pool in Azure Synapse Analytics?

- Azure HDInsight.
- Apache Spark Console.
- Spark Instance.

Explanation

Correct. The definition of an Apache Spark pool is that, when instantiated, it is used to create an Apache Spark instance that processes data.

Question 2

In order to create an Apache Spark pool in Azure Synapse Analytics, what needs to be created first?

- Azure Synapse Analytics Workspace.
- Azure Synapse Studio.
- An Apache Spark Instance.

Explanation

Correct. In order to create an Apache Spark pool in Azure Synapse Analytics, you would have to create a Synapse Analytics Workspace.

Question 1

What are Azure Synapse Studio notebooks based on?

- dedicated SQL pool.
- Apache Spark.
- Apache Spark pool.

Explanation

Correct. Azure Synapse Studio notebook is purely Apache Spark based.

Question 2

How can all notebooks in Synapse studio be saved?

- Select the **Publish all** button on the workspace command bar.
- Select the **Publish** button on the notebook command bar.
- Using CTRL + S.

Explanation

*Correct. To save all notebooks in your workspace, select the **Publish all** button on the workspace command bar.*

Question 1

What is a step in flattening a nested schema?

- Create a parquet file.
- Explode Arrays.
- Load a csv file.

Explanation

Correct. Explode Arrays is a third step in flattening nested schemas. It is necessary to transform the array in the DataFrame into a new DataFrame where the column that you want to select is defined.

Question 2

What is a dataframe?

- A creation of a data structure.
- A parquet file.
- A csv file.

Explanation

Correct. A DataFrame creates a data structure and it's one of the core data structures in Apache Spark.

Question 1

In what language can the Azure Synapse Apache Spark to Synapse SQL connector can be used?

- Python.
- SQL.
- Scala.

Explanation

Correct. The connector uses Scala to integrate Apache Spark pools with dedicated SQL pools in Azure Synapse Analytics.

Question 2

When is it unnecessary to use import statements for transferring data between a dedicated SQL and Apache Spark pool?

- Use the integrated notebook experience from Azure Synapse Studio.
- Use the PySpark connector.
- Use token-based authentication.

Explanation

Correct. Import statements are not needed since they are pre-loaded in case you use the Azure Synapse Studio integrated notebook experience.

Question 1

What is one of the possible ways to optimize an Apache Spark Job?

- Remove all nodes.
- Remove the Apache Spark Pool.
- Use bucketing.

Explanation

Correct. Bucketed tables are optimized because it is a metadata operation about how the data is bucketed and sorted.

Question 2

What can cause a slower performance on join or shuffle jobs?

- Data skew.
- Enablement of autoscaling
- Bucketing.

Explanation

Correct. Due to asymmetry in your job data.

Module 5 Ingest and load Data into the Data Warehouse

Module introduction

module-introduction

In this module, you will learn how to use the best practices you need to adopt to load data into a data warehouse in Azure Synapse Analytics. You will understand the various methods that can be used to ingest data between various data stores using Azure Data Factory or Azure Synapse Pipelines. You will also perform petabyte-scale ingestion with Azure Data Factory or Azure Synapse Pipelines.

Learning objectives

In this module, you will:

- Use data loading best practices in Azure Synapse Analytics
- Perform Petabyte-scale ingestion with Azure Data Factory or Azure Synapse Pipelines

Use data loading best practices in Azure Synapse Analytics

Introduction

Learn about data loading techniques for Azure Synapse Analytics SQL Pools for use in analytical workloads.

In this module, you will:

- Understand data loading design goals
- Explain loading methods into Azure Synapse Analytics
- Manage source data files
- Manage singleton updates
- Set-up dedicated data loading accounts
- Manage concurrent access to Azure Synapse Analytics
- Implement Workload Management
- Simplify ingestion with the Copy Activity

Prerequisites

Before taking this module, it is recommended that the student is able to:

- Log into the Azure portal
- Create a Synapse Analytics Workspace
- Create an Azure Synapse Analytics SQL Pool

Understand data load design goals

Optimizing and speeding up data loads to minimize the impact on the performance of ongoing queries is a key design goal in data warehousing.

Analytical systems are constantly balanced between loading and querying workloads. Some analytical systems have loading requirements that require data to be available in near real-time, others periodically throughout the business day, or at the end of the month. Most systems you will find have a mixture of these dependants on the data sources being ingested, and type of work being done with the data.

Loading data is essential because of the need to query or analyze the data to gain insights from it, so one of the main design goals in loading data, is to manage or minimize the impact on analytical workloads while loading the data with the highest throughput possible.

Some of the questions that need to be asked when considering data loading include:

- Where is my data coming from?
- Is the data net new? or do you receive changes from existing datasets?
- How often is the data being refreshed, added to or replaced?
- What formats are the data coming in?
- Is the data ingestible as-is? or are transformations and cleansing tasks required?

- Should I transform the data prior to or after loading?
- How complex or robust does the loading process have to be?
- Which takes priority, loading or querying/analysis?

By answering these questions, you'll understand the design goals and considerations to the process of designing appropriate workflows and pipelines and which tools and methods you'll need to utilize.

Explain load methods into Azure Synapse Analytics

Azure Synapse Analytics has a rich set of tools and methods available to load data into SQL Pools. You can load data from relational or non-relational datastores; structured or semi-structured; on premises systems or other clouds; in batches or streams.

For the purposes of this module, we'll consider data stored in structured and semi structured formats in Azure Blob Storage or Azure Data Lake Store. Based on the variety of data that you work with, your data loads can include:

Data loads directly from Azure storage with transact-sql and the copy statement

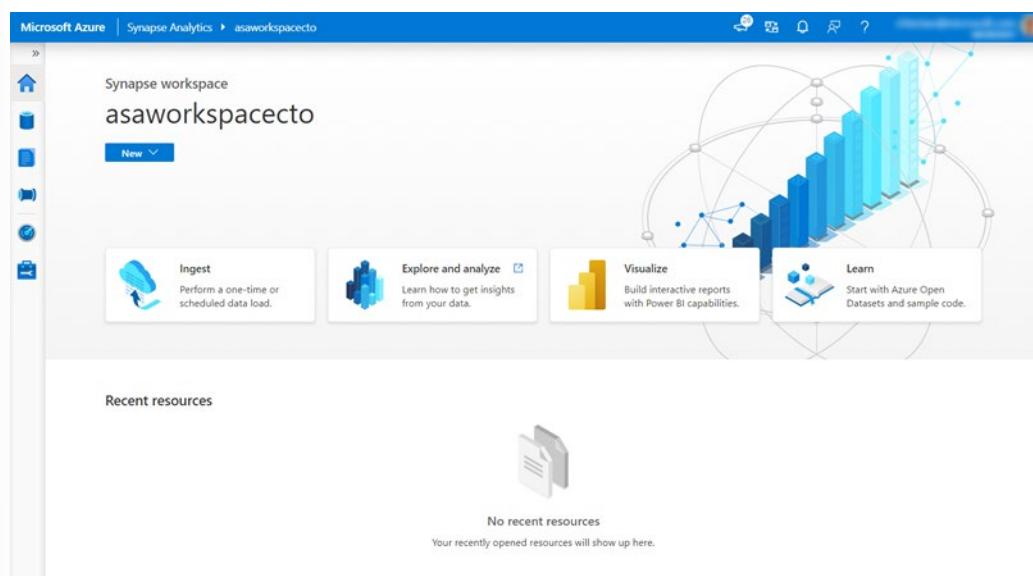
Within Azure Synapse Studio, you can write Transact-SQL code that runs against any configured SQL Pools within the workspace. Similarly, within the same Transact-SQL script, you can read and digest data from Azure Blob Storage or Azure Data Lake and insert it into a table within the SQL Pool

Perform data loads using Azure synapse pipeline data flows.

Data flows are a key feature within the Azure Synapse Studio experience. You can access the data flows from the Integrate hub. From within the Develop hub, you're able to access configured source repositories and run transformations against them to a variety of destinations referred to as sinks.

Use polybase by defining external tables

Using Transact-SQL, you can use PolyBase to access files that are located directly on Azure Storage as if they were structured tables within your SQL Pool. You define an **external data source** pointing to the location of the file or the folder the files reside in, the external file format, which can be GZip compressed delimited text, ORC, Parquet or JSON, and then the external table with the column attributes that map to the structure from the external files.



Manage source data files

When loading data into Azure Synapse Analytics on a scheduled basis, it's important to try to reduce the time taken to not perform the data load, and minimize the resources needed as much as possible to maintain good performance cost-effectively.

Strategies for managing source data files include:

Maintain a well-engineered data lake structure

Maintaining a well-engineered Data Lake structure allows you to know that the data you're loading regularly is consistent with the data requirements for your system. It is less important if your load is a once-off or exploratory rather than analytical. Some strategies include folder hierarchies based on the source system, and date/time or file format and focus.

In general, having well defined "zones" established for the data coming into the Data Lake and cleansing and transformation tasks that land the data you need in a curated and optimized state.

Compress and optimize files

When loading large datasets, it's best to use the compression capabilities of the file format. It ensures that less time is spent on the process of data transfers, using instead the power of Azure Synapse' Massively Parallel Processing (MPP) compute capabilities for decompression

It is fairly standard to maintain curated source files in columnar compressed file formats such as RC, Gzip, Parquet, and ORC, which are all supported import formats.

Split source files

One of the key architectural components within Azure Synapse Analytics dedicated SQL pools is the decoupled storage that is segmented into 60 parts. You should maintain alignment to multiples of this number as much as possible depending on the file sizes that you are loading, and the number of compute nodes you have provisioned. Since there are 60 storage segments and a maximum of 60 MPP

compute nodes within the highest performance configuration of SQL Pools, a 1:1 file to compute node to storage segment may be viable for ultra-high workloads, reducing the load times to the minimum possible.

Manage singleton updates

A data warehouse that is built on a Massively Parallel Processing (MPP) system are built for processing and analyzing large datasets. As such they perform well with larger batch type loads and updates that can be distributed across the compute nodes and storage.

Singleton or smaller transaction batch loads should be grouped into larger batches to optimize the Synapse SQL Pools processing capabilities. To be clear, A one-off load to a small table with an INSERT statement may be the best approach, if it is a one-off.

However, if you need to load thousands or millions of rows throughout the day, then singleton INSERTS aren't optimal against an MPP system. One way to solve this issue is to develop one process that writes the outputs of an INSERT statement to a file, and then another process to periodically load this file to take advantage of the parallelism that Azure Synapse Analytics.

Set-up dedicated data load accounts

A mistake that many people make when first exploring dedicated SQL Pools are to use the service administrator account as the one used for loading data. This account is limited to using the smallrc dynamic resource class that can use between 3% and 25% of the resources depending on the performance level of the provisioned SQL Pools.

Instead, it's better to create specific accounts assigned to different resource classes dependent on the anticipated task. This will optimize load performance and maintain concurrency as required by managing the available resource slots available within the dedicated SQL Pool.

This example creates a loading user classified to a specific workload group.

1. The first step is to connect to master and create a login.

```
-- Connect to master  
CREATE LOGIN loader WITH PASSWORD = 'a123STRONGpassword!';
```

2. Next, connect to the dedicated SQL pool and create a user.

The following code assumes you're connected to the database called mySampleDataWarehouse. It shows how to create a user called loader and gives the user permissions to create tables and load using the COPY statement. Then it classifies the user to the DataLoads workload group with maximum resources.

```
-- Connect to the SQL pool  
CREATE USER loader FOR LOGIN loader;  
GRANT ADMINISTER DATABASE BULK OPERATIONS TO loader;  
GRANT INSERT ON <yourtablename> TO loader;  
GRANT SELECT ON <yourtablename> TO loader;  
GRANT CREATE TABLE TO loader;  
GRANT ALTER ON SCHEMA::dbo TO loader;  
  
CREATE WORKLOAD GROUP DataLoads  
WITH (  
    MIN_PERCENTAGE_RESOURCE = 100
```

```
,CAP_PERCENTAGE_RESOURCE = 100  
,REQUEST_MIN_RESOURCE_GRANT_PERCENT = 100  
);  
  
CREATE WORKLOAD CLASSIFIER [wgcELTLogin]  
WITH (  
    WORKLOAD_GROUP = 'DataLoads'  
    ,MEMBERNAME = 'loader'  
);
```

Manage concurrent access to Azure Synapse Analytics

Concurrency and the allocation of resources across connected users are also a factor that can limit the load performance into Azure Synapse Analytics SQL pools.

SQL Pools have the concept of concurrency slots, which manage the allocation of memory to connected users. To optimize the load execution operations, consider reducing or minimizing the number of simultaneous load jobs that are running or assigning higher resource classes that reduce the number of active running tasks.

Implement workload management

Azure Synapse Analytics allows you to create, control, and manage resource availability when workloads are competing. This allows you to manage the relative importance of each workload when waiting for available resources.

To facilitate faster load times, you can create a workload classifier for the load user with the "importance" set to above_normal or High. Workload importance ensures that the load takes precedence over other waiting tasks of a lower importance rating. Use this in conjunction with your own workload group definitions for workload isolation to manage minimum and maximum resource allocations during peak and quiet periods.

Dedicated SQL pool workload management in Azure Synapse consists of three high-level concepts:

- Workload Classification
- Workload Importance
- Workload Isolation

These capabilities give you more control over how your workload utilizes system resources.

Workload classification

Workload management classification allows workload policies to be applied to requests through assigning resource classes and importance.

While there are many ways to classify data warehousing workloads, the simplest and most common classification is load and query. You load data with insert, update, and delete statements. You query the data using selects. A data warehousing solution will often have a workload policy for load activity, such as assigning a higher resource class with more resources. A different workload policy could apply to queries, such as lower importance compared to load activities.

You can also subclassify your load and query workloads. Subclassification gives you more control of your workloads. For example, query workloads can consist of cube refreshes, dashboard queries or ad-hoc queries. You can classify each of these query workloads with different resource classes or importance settings. Load can also benefit from subclassification. Large transformations can be assigned to larger resource classes. Higher importance can be used to ensure key sales data is loaded before weather data or a social data feed.

Not all statements are classified as they do not require resources or need importance to influence execution. DBCC commands, BEGIN, COMMIT, and ROLLBACK TRANSACTION statements are not classified.

Workload importance

Workload importance influences the order in which a request gets access to resources. On a busy system, a request with higher importance has first access to resources. Importance can also ensure ordered access to locks. There are five levels of importance: low, below_normal, normal, above_normal, and high. Requests that don't set importance are assigned the default level of normal. Requests that have the same importance level have the same scheduling behavior that exists today.

Workload isolation

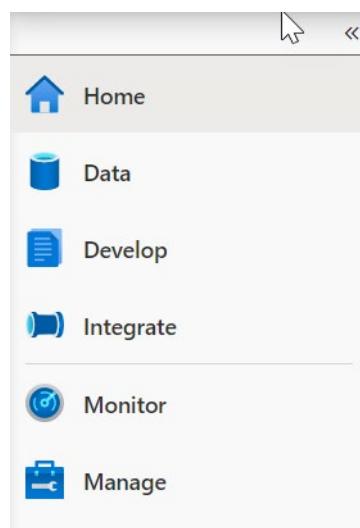
Workload isolation reserves resources for a workload group. Resources reserved in a workload group are held exclusively for that workload group to ensure execution. Workload groups also allow you to define the amount of resources that are assigned per request, much like resource classes do. Workload groups give you the ability to reserve or cap the amount of resources a set of requests can consume. Finally, workload groups are a mechanism to apply rules, such as query timeout, to requests.

Exercise - implement workload management

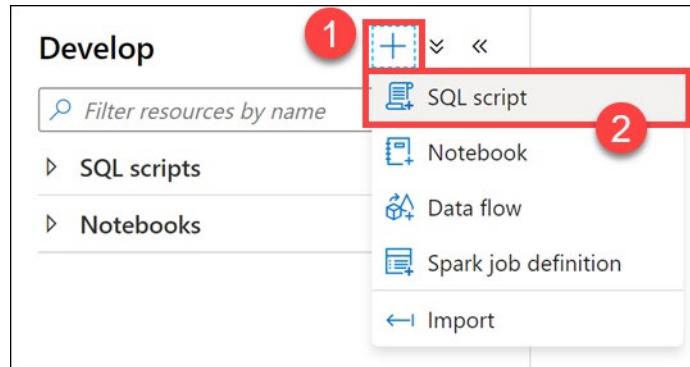
Create a workload classifier to add importance to certain queries

Your organization has asked you if there is a way to mark queries executed by the CEO as more important than others, so they don't appear slow due to heavy data loading or other workloads in the queue. You decide to create a workload classifier and add importance to prioritize the CEO's queries.

1. Select the **Develop** hub.



- From the **Develop** menu, select the + button (1) and choose **SQL Script** (2) from the context menu.



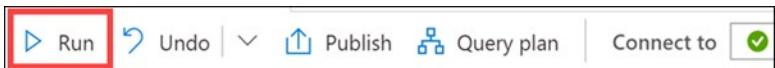
- In the toolbar menu, connect to the **SQL Pool** database to execute the query.



- In the query window, replace the script with the following to confirm that there are no queries currently being run by users logged in as `asa.sql.workload01`, representing the CEO of the organization or `asa.sql.workload02` representing the data analyst working on the project:
--First, let's confirm that there are no queries currently being run by users logged in workload01 or workload02

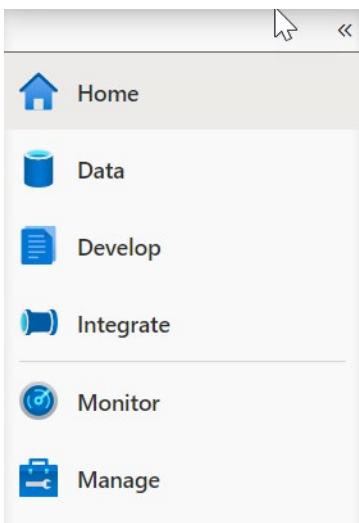
```
SELECT s.login_name, r.[Status], r.Importance, submit_time,
start_time ,s.session_id FROM sys.dm_pdw_exec_sessions s
JOIN sys.dm_pdw_exec_requests r ON s.session_id = r.session_id
WHERE s.login_name IN ('asa.sql.workload01','asa.sql.workload02') and Importance
is not NULL AND r.[status] in ('Running','Suspended')
--and submit_time>dateadd(minute,-2,getdate())
ORDER BY submit_time ,s.login_name
```

- Select **Run** from the toolbar menu to execute the SQL command.

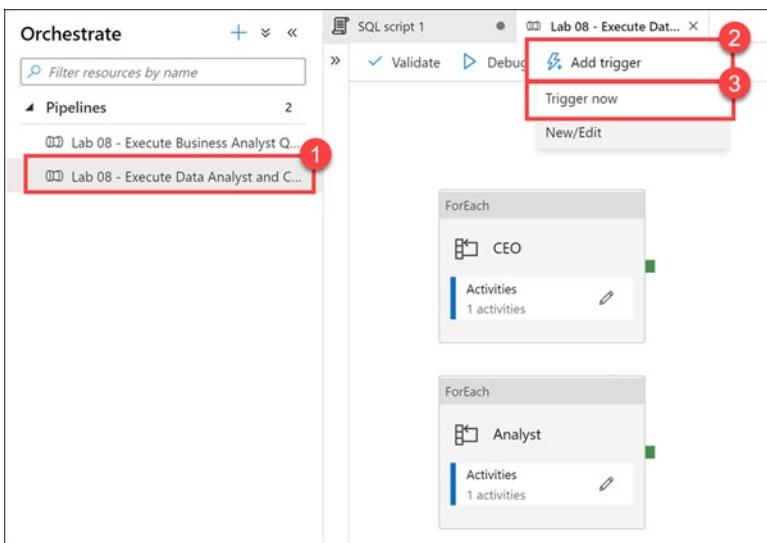


Now that we have confirmed that there are no running queries, we need to flood the system with queries and see what happens for `asa.sql.workload01` and `asa.sql.workload02`. To do this, we'll run a Azure Synapse Pipeline which triggers queries.

6. Select the **Integrate** hub.



7. Select the **Lab 08 - Execute Data Analyst and CEO Queries** Pipeline (1), which will run / trigger the `asa.sql.workload01` and `asa.sql.workload02` queries. Select **Add trigger** (2), then **Trigger now** (3). In the dialog that appears, select **OK**.



8. Let's see what happened to all the queries we just triggered as they flood the system. In the query window, replace the script with the following:

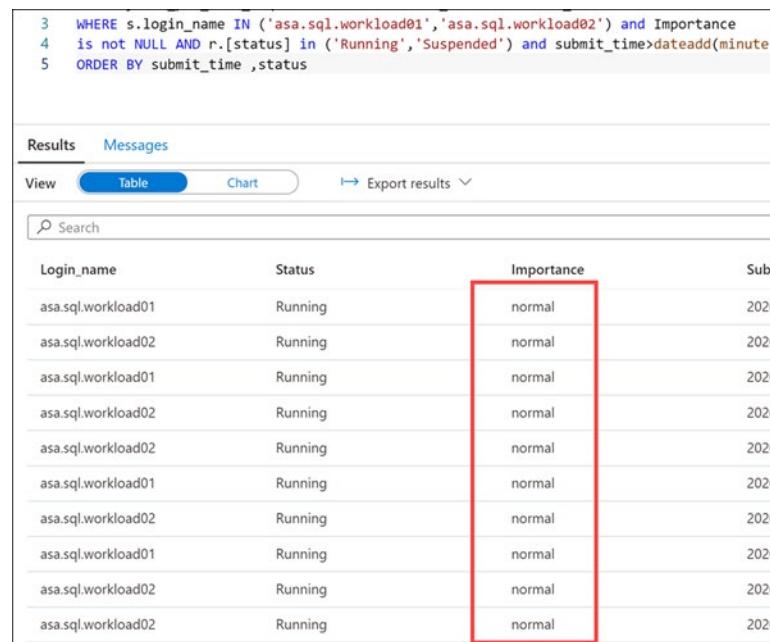
```
SELECT s.login_name, r.[Status], r.Importance, submit_time, start_time ,s.session_id FROM sys.dm_pdw_exec_sessions s
JOIN sys.dm_pdw_exec_requests r ON s.session_id = r.session_id
```

```
WHERE s.login_name IN ('asa.sql.workload01','asa.sql.workload02') and Importance  
is not NULL AND r.[status] in ('Running','Suspended') and submit_time>dateadd(minute,-2,getdate())  
ORDER BY submit_time ,status
```

9. Select **Run** from the toolbar menu to execute the SQL command.



You should see an output similar to the following:



Login_name	Status	Importance	Subm
asa.sql.workload01	Running	normal	2020-
asa.sql.workload02	Running	normal	2020-
asa.sql.workload01	Running	normal	2020-
asa.sql.workload02	Running	normal	2020-
asa.sql.workload02	Running	normal	2020-
asa.sql.workload01	Running	normal	2020-
asa.sql.workload02	Running	normal	2020-
asa.sql.workload01	Running	normal	2020-
asa.sql.workload02	Running	normal	2020-
asa.sql.workload02	Running	normal	2020-

Notice that the **Importance** level for all queries is set to **normal**.

10. We will give our `asa.sql.workload01` user queries priority by implementing the **Workload Importance** feature. In the query window, replace the script with the following:

```
IF EXISTS (SELECT * FROM sys.workload_management_workload_classifiers WHERE  
name = 'CEO')  
BEGIN  
    DROP WORKLOAD CLASSIFIER CEO;  
END  
CREATE WORKLOAD CLASSIFIER CEO  
WITH (WORKLOAD_GROUP = 'largerc'  
, MEMBERNAME = 'asa.sql.workload01', IMPORTANCE = High);
```

We are executing this script to create a new **Workload Classifier** named `CEO` that uses the `largerc` Workload Group and sets the **Importance** level of the queries to **High**.

11. Select **Run** from the toolbar menu to execute the SQL command.

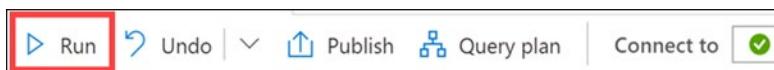


12. Let's flood the system again with queries and see what happens this time for asa.sql.workload01 and asa.sql.workload02 queries. To do this, we'll run an Azure Synapse Pipeline which triggers queries. **Select** the Integrate Tab, **run** the **Lab 08 - Execute Data Analyst and CEO Queries** Pipeline, which will run / trigger the asa.sql.workload01 and asa.sql.workload02 queries.

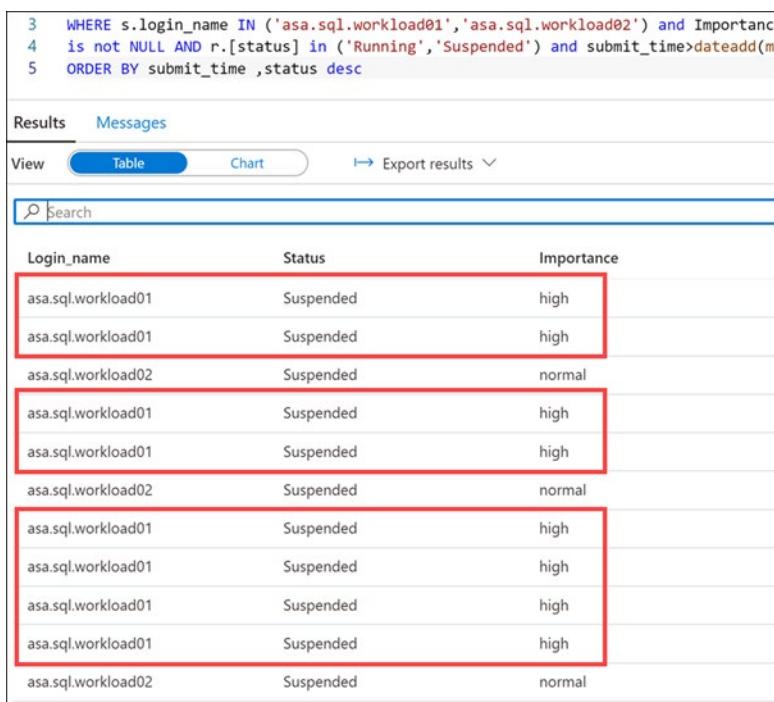
13. In the query window, replace the script with the following to see what happens to the asa.sql.workload01 queries this time:

```
SELECT s.login_name, r.[Status], r.Importance, submit_time, start_time ,s.session_id FROM sys.dm_pdw_exec_sessions s  
JOIN sys.dm_pdw_exec_requests r ON s.session_id = r.session_id  
WHERE s.login_name IN ('asa.sql.workload01','asa.sql.workload02') and Importance  
is not NULL AND r.[status] in ('Running','Suspended') and submit_time>dateadd(minute,-2,getdate())  
ORDER BY submit_time ,status desc
```

14. Select **Run** from the toolbar menu to execute the SQL command.



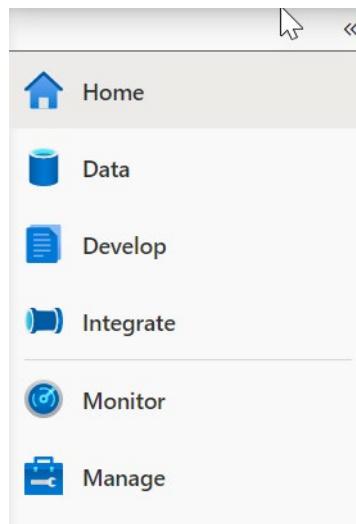
You should see an output similar to the following:



3 WHERE s.login_name IN ('asa.sql.workload01','asa.sql.workload02') and Importance		
4 is not NULL AND r.[status] in ('Running','Suspended') and submit_time>dateadd(m		
5 ORDER BY submit_time ,status desc		
Results Messages		
View Table Chart Export results		
Search		
Login_name	Status	Importance
asa.sql.workload01	Suspended	high
asa.sql.workload01	Suspended	high
asa.sql.workload02	Suspended	normal
asa.sql.workload01	Suspended	high
asa.sql.workload01	Suspended	high
asa.sql.workload02	Suspended	normal
asa.sql.workload01	Suspended	high
asa.sql.workload01	Suspended	high
asa.sql.workload01	Suspended	high
asa.sql.workload02	Suspended	normal

Notice that the queries executed by the asa.sql.workload01 user have a **high** importance.

15. Select the **Monitor** hub.



16. Select **Pipeline runs** (1), and then select **Cancel recursive** (2) for each running Lab 08 pipelines, marked **In progress** (3). This will help speed up the remaining tasks.

Pipeline runs					
	Run start ↑	Run end	Duration	Triggered by	Status
<input type="checkbox"/> Pipeline name	9/8/20, 11:19:04 PM	--	00:13:47	Manual trigger	
<input type="checkbox"/> Lab 08 - Execute D...	9/8/20, 11:19:04 PM	--	00:13:47	Manual trigger	
<input type="checkbox"/> Lab 08 - Execute Data Analyst ...	9/8/20, 11:19:04 PM	--	00:13:47	Manual trigger	
<input type="checkbox"/> Lab 08 - Execute Data Analyst ...	9/8/20, 11:19:04 PM	--	00:13:47	Manual trigger	
<input type="checkbox"/> Lab 08 - Execute Data Analyst ...	9/8/20, 11:19:04 PM	--	00:13:47	Manual trigger	
<input type="checkbox"/> Setup - Load SQL Pool	9/8/20, 2:49:12 PM	9/8/20, 2:49:47 PM	00:00:34	Manual trigger	
<input type="checkbox"/> Setup - Load SQL Pool (global)	9/8/20, 1:54:04 PM	9/8/20, 2:33:05 PM	00:39:01	Manual trigger	

Reserve resources for specific workloads through workload isolation

Workload isolation means resources are reserved, exclusively, for a workload group. Workload groups are containers for a set of requests and are the basis for how workload management, including workload isolation, is configured on a system. A simple workload management configuration can manage data loads and user queries.

In the absence of workload isolation, requests operate in the shared pool of resources. Access to resources in the shared pool is not guaranteed and is assigned on an importance basis.

Given the workload requirements provided by Tailwind Traders, you decide to create a new workload group called `CEO` to reserve resources for queries executed by the CEO.

Let's start by experimenting with different parameters.

1. In the query window, replace the script with the following:

```
IF NOT EXISTS (SELECT * FROM sys.workload_management_workload_groups where name = 'CEO')
BEGIN
    Create WORKLOAD GROUP CEO WITH
```

```
( MIN_PERCENTAGE_RESOURCE = 50      -- integer value  
,REQUEST_MIN_RESOURCE_GRANT_PERCENT = 25 --  
,CAP_PERCENTAGE_RESOURCE = 100  
)  
END
```

The script creates a workload group called `CEODemo` to reserve resources exclusively for the workload group. In this example, a workload group with a `MIN_PERCENTAGE_RESOURCE` set to 50% and `REQUEST_MIN_RESOURCE_GRANT_PERCENT` set to 25% is guaranteed 2 concurrency.

2. Select **Run** from the toolbar menu to execute the SQL command.

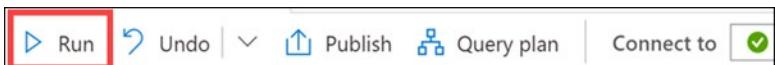


3. In the query window, replace the script with the following to create a Workload Classifier called `CEODreamDemo` that assigns a workload group and importance to incoming requests:

```
IF NOT EXISTS (SELECT * FROM sys.workload_management_workload_classifiers where name =  
'CEODreamDemo')  
BEGIN  
    Create Workload Classifier CEODreamDemo with  
    ( Workload_Group ='CEODemo',MemberName='asa.sql.workload02',IMPORTANCE = BELOW_NOR-  
MAL);  
END
```

This script sets the Importance to `BELOW_NORMAL` for the `asa.sql.workload02` user, through the new `CEODreamDemo` Workload Classifier.

4. Select **Run** from the toolbar menu to execute the SQL command.



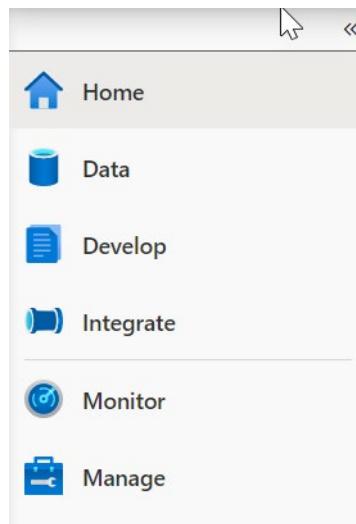
5. In the query window, replace the script with the following to confirm that there are no active queries being run by `asa.sql.workload02` (suspended queries are OK):

```
SELECT s.login_name, r.[Status], r.Importance, submit_time,  
start_time ,s.session_id FROM sys.dm_pdw_exec_sessions s  
JOIN sys.dm_pdw_exec_requests r ON s.session_id = r.session_id  
WHERE s.login_name IN ('asa.sql.workload02') and Importance  
is not NULL AND r.[status] in ('Running','Suspended')  
ORDER BY submit_time, status
```

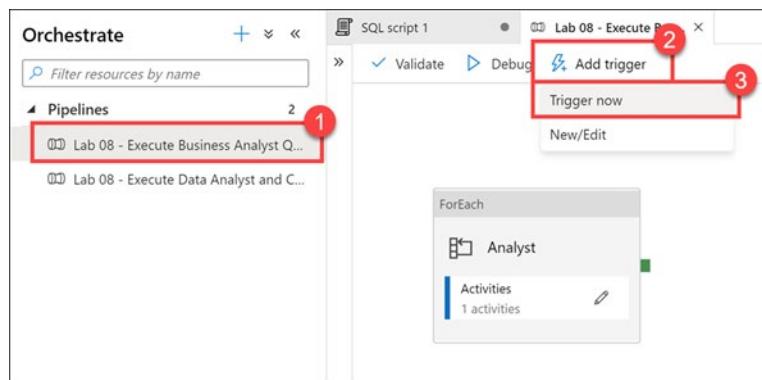
6. Select **Run** from the toolbar menu to execute the SQL command.



7. Select the **Integrate** hub.



8. Select the **Lab 08 - Execute Business Analyst Queries** Pipeline (1), which will run / trigger `asa.sql.workload02` queries. Select **Add trigger** (2), then **Trigger now** (3). In the dialog that appears, select **OK**.



9. In the query window, replace the script with the following to see what happened to all the `asa.sql.workload02` queries we just triggered as they flood the system:

```
SELECT s.login_name, r.[Status], r.Importance, submit_time,
start_time ,s.session_id FROM sys.dm_pdw_exec_sessions s
JOIN sys.dm_pdw_exec_requests r ON s.session_id = r.session_id
WHERE s.login_name IN ('asa.sql.workload02') and Importance
is not NULL AND r.[status] in ('Running','Suspended')
ORDER BY submit_time, status
```

10. Select **Run** from the toolbar menu to execute the SQL command.

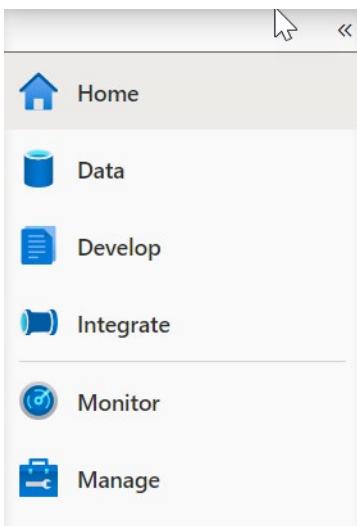


You should see an output similar to the following that shows the importance for each session set to `below_normal`:

4 WHERE s.login_name IN ('asa.sql.workload02') and Importance 5 is not NULL AND r.[status] in ('Running', 'Suspended') 6 ORDER BY submit_time, status	
Results Messages	
View Table Chart Export results	
Search	
Login_name Status Importance Submit_time	
asa.sql.workload02 Running below_normal 2020-09-08T11:39:04Z	1
asa.sql.workload02 Running below_normal 2020-09-08T11:39:04Z	2
asa.sql.workload02 Running below_normal 2020-09-08T11:39:04Z	
asa.sql.workload02 Running below_normal 2020-09-08T11:39:04Z	

Notice that the running scripts are executed by the `asa.sql.workload02` user (1) with an Importance level of **below_normal** (2). We have successfully configured the business analyst queries to execute at a lower importance than the CEO queries. We can also see that the `CEODreamDemo` Workload Classifier works as expected.

11. Select the **Monitor** hub.



12. Select **Pipeline runs** (1), and then select **Cancel recursive** (2) for each running Lab 08 pipelines, marked **In progress** (3). This will help speed up the remaining tasks.

Orchestration		Pipeline runs				
	1 Pipeline runs	Triggered	Debug	Rerun	Cancel	Refresh
	2 Trigger runs					
	3 Integration runtimes					
	Activities					
	4 Apache Spark applications					
	5 SQL requests					
	6 Data flow debug					
		Showing 1 - 10 items	Run start	Run end	Duration	Triggered by
		<input type="checkbox"/> Pipeline name	2 9/9/20, 10:03:02 AM	--	00:05:24	Manual trigger
		<input type="checkbox"/> Lab 08 - Execute B... 2	9/9/20, 10:03:02 AM	Cancel recursive		In progress
		<input type="checkbox"/> Lab 08 - Execute Data Analyst... 2	9/8/20, 11:39:04 PM	9/8/20, 11:39:22 PM	00:20:18	Manual trigger
		<input type="checkbox"/> Lab 08 - Execute Data Analyst ... 2	9/8/20, 11:39:15 PM	9/8/20, 11:39:19 PM	00:22:44	Manual trigger
		Status				
		2 In progress				
		2 Cancelled				
		2 Cancelled				

13. Return to the query window under the **Develop** hub. In the query window, replace the script with the following to set 3.25% minimum resources per request:

```
IF EXISTS (SELECT * FROM sys.workload_management_workload_classifiers where group_name = 'CEO Demo')
BEGIN
    Drop Workload Classifier CEO Dream Demo
    DROP WORKLOAD GROUP CEO Demo
    --- Creates a workload group 'CEO Demo'.
    Create WORKLOAD GROUP CEO Demo WITH
        (MIN_PERCENTAGE_RESOURCE = 26 -- integer value
         ,REQUEST_MIN_RESOURCE_GRANT_PERCENT = 3.25 -- factor of 26 (guaranteed more than 4
concurrents)
         ,CAP_PERCENTAGE_RESOURCE = 100
        )
    --- Creates a workload Classifier 'CEO Dream Demo'.
    Create Workload Classifier CEO Dream Demo with
        (Workload_Group = 'CEO Demo', MemberName = 'asa.sql.workload02', IMPORTANCE = BELOW_NORMAL);
END
```

NOTE: Configuring workload containment implicitly defines a maximum level of concurrency. With a CAP_PERCENTAGE_RESOURCE set to 60% and a REQUEST_MIN_RESOURCE_GRANT_PERCENT set to 1%, up to a 60-concurrency level is allowed for the workload group. Consider the method included below for determining the maximum concurrency:

[Max Concurrency] = [CAP_PERCENTAGE_RESOURCE] / [REQUEST_MIN_RESOURCE_GRANT_PERCENT]

14. Select **Run** from the toolbar menu to execute the SQL command.



15. Let's flood the system again and see what happens for `asa.sql.workload02`. To do this, we will run an Azure Synapse Pipeline which triggers queries. Select the **Integrate Tab**. **Run** the **Lab 08 - Execute Business Analyst Queries** Pipeline, which will run / trigger `asa.sql.workload02` queries.

16. In the query window, replace the script with the following to see what happened to all of the `asa.sql.workload02` queries we just triggered as they flood the system:

```
SELECT s.login_name, r.[Status], r.Importance, submit_time,
start_time, s.session_id FROM sys.dm_pdw_exec_sessions s
JOIN sys.dm_pdw_exec_requests r ON s.session_id = r.session_id
WHERE s.login_name IN ('asa.sql.workload02') and Importance
is not NULL AND r.[status] in ('Running','Suspended')
ORDER BY submit_time, status
```

17. Select **Run** from the toolbar menu to execute the SQL command.



After several moments (up to a minute), we should see several concurrent executions by the `asa.sql.workload02` user running at **below_normal** importance. We have validated that the modified Workload Group and Workload Classifier works as expected.

```

4 WHERE s.login_name IN ('asa.sql.workload02') and Importance
5 is not NULL AND r.[status] in ('Running','Suspended')
6 ORDER BY submit_time, status

```

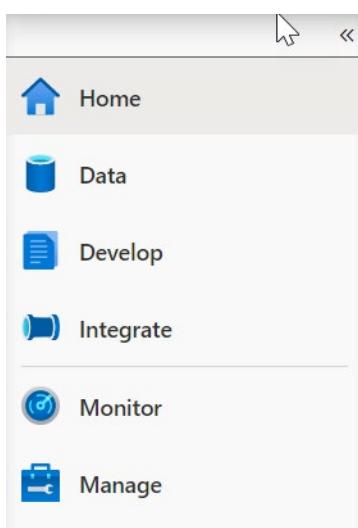
Results **Messages**

View **Table** **Chart** **Export results**

Search

Login_name	Status	Importance
asa.sql.workload02	Running	below_normal

18. Select the **Monitor** hub.



19. Select **Pipeline runs** (1), and then select **Cancel recursive** (2) for each running Lab 08 pipelines, marked **In progress** (3). This will help speed up the remaining tasks.

The screenshot shows the 'Pipeline runs' page in the Azure portal. The left sidebar has a red box around the 'Orchestration' section, with a red circle containing the number '1' above the 'Pipeline runs' item. The main area displays a table of pipeline runs. One run is highlighted with a red box and a red circle containing the number '2', with a red arrow pointing to the 'Cancel recursive' button. Another run is highlighted with a red box and a red circle containing the number '3', with a red arrow pointing to the 'Status' column showing 'In progress'.

Pipeline name	Run start	Run end	Duration	Triggered by	Status
Lab 08 - Execute B...	9/9/20, 10:03:02 AM	--	00:05:24	Manual trigger	In progress
Lab 08 - Execute Data Analyst...	9/8/20, 11:39:04 PM	9/8/20, 11:39:22 PM	00:20:18	Manual trigger	Cancelled
Lab 08 - Execute Data Analyst ...	9/8/20, 11:16:35 PM	9/8/20, 11:39:19 PM	00:22:44	Manual trigger	Cancelled

Simplify ingestion with the Copy Activity

The broad capabilities of the Copy Activity allow you to quickly and easily move data into SQL Pools from a variety of sources.

There are different options for loading large amounts and varying types of data into Azure Synapse Analytics, such as through T-SQL commands using a dedicated SQL pool, and with Azure Synapse pipelines. There are some things to consider when loading large or disparate file types and formats. When files are stored in ADLS Gen2, you can use either PolyBase external tables or the new COPY statement when using T-SQL commands. Both options enable fast and scalable data load operations, but there are some differences between the two:

PolyBase	COPY
Needs CONTROL permission	Relaxed permission
Has row width limits	No row width limit
No delimiters within text	Supports delimiters in text
Fixed line delimiter	Supports custom column and row delimiters
Complex to set up in code	Reduces amount of code

Ingest data using PolyBase

PolyBase requires the following elements:

- An external data source that points to the `abfss` path in ADLS Gen2 where the Parquet files are located
 - An external file format for Parquet files
 - An external table that defines the schema for the files, as well as the location, data source, and file format
1. Create a new SQL script with the following to create the external data source. Be sure to replace `YOURACCOUNT` with the name of your ADLS Gen2 account:

```
-- Replace YOURACCOUNT with the name of your ADLS Gen2 account.  
CREATE EXTERNAL DATA SOURCE ABSS  
WITH  
( TYPE = HADOOP,  
    LOCATION = 'abfss://wwi-02@YOURACCOUNT.dfs.core.windows.net'  
);
```

2. Select **Run** from the toolbar menu to execute the SQL command.
3. In the query window, replace the script with the following to create the external file format and external data table. Notice that we defined `TransactionId` as an `nvarchar(36)` field instead of `uniqueidentifier`. This is because external tables do not currently support `uniqueidentifier` columns:

```
CREATE EXTERNAL FILE FORMAT [ParquetFormat]  
WITH (  
    FORMAT_TYPE = PARQUET,  
    DATA_COMPRESSION = 'org.apache.hadoop.io.compress.SnappyCodec'  
)  
GO
```

```
CREATE SCHEMA [wwi_external];
GO

CREATE EXTERNAL TABLE [wwi_external].Sales
(
    [TransactionId] <a href="36" title="" target="_blank" data-generated="">nvarchar</a> NOT
NULL,
    [CustomerId] [int] NOT NULL,
    [ProductId] [smallint] NOT NULL,
    [Quantity] [smallint] NOT NULL,
    [Price] <a href="9,2" title="" target="_blank" data-generated="">decimal</a> NOT NULL,
    [TotalAmount] <a href="9,2" title="" target="_blank" data-generated="">decimal</a> NOT
NULL,
    [TransactionDate] [int] NOT NULL,
    [ProfitAmount] <a href="9,2" title="" target="_blank" data-generated="">decimal</a> NOT
NULL,
    [Hour] [tinyint] NOT NULL,
    [Minute] [tinyint] NOT NULL,
    [StoreId] [smallint] NOT NULL
)
WITH
(
    LOCATION = '/sale-small%2FYear%3D2019',
    DATA_SOURCE = ABSS,
    FILE_FORMAT = [ParquetFormat]
)
GO
```

Note: The /sale-small/Year=2019/ folder's Parquet files contain **339,507,246 rows**.

4. Select **Run** from the toolbar menu to execute the SQL command.
5. In the query window, replace the script with the following to load the data into the `wwi_staging.SalesHeap` table. **DO NOT RUN** this command. In the interest of time, we will skip this command since it takes around 6 minutes to execute:

```
INSERT INTO [wwi_staging].[SaleHeap]
SELECT *
FROM [wwi_external].[Sales]
```

Simplify ingestion with the COPY activity

Now let's see how to perform the same load operation with the COPY statement.

1. In the query window, replace the script with the following to truncate the heap table and load data using the COPY statement. As you did before, be sure to replace `YOURACCOUNT` with the name of your ADLS Gen2 account:

```
TRUNCATE TABLE wwi_staging.SaleHeap;
GO
```

```
-- Replace YOURACCOUNT with the workspace default storage account name.  
COPY INTO wwi_staging.SaleHeap  
FROM 'https://YOURACCOUNT.dfs.core.windows.net/wwi-02/sale-small%2FYear%3D2019'  
WITH (  
    FILE_TYPE = 'PARQUET',  
    COMPRESSION = 'SNAPPY'  
)  
GO
```

2. Select **Run** from the toolbar menu to execute the SQL command. It takes a few minutes to execute this command. **Take note** of how long it took to execute this query.

3. In the query window, replace the script with the following to see how many rows were imported:

```
SELECT COUNT(1) FROM wwi_staging.SaleHeap(nolock)
```

4. Select **Run** from the toolbar menu to execute the SQL command. You should see a result of 339507246.

Do the number of rows match for both load operations? Which activity was fastest? You should see that both copied the same amount of data in roughly the same amount of time.

More COPY activity examples

The Copy Activity supports a large range of data sources and sinks on-premises and in the cloud. It facilitates the efficient, yet flexible parsing and transfer of data or files between systems in an optimized fashion as well as giving you capability of easily converting datasets into other formats.

In the following example, you can load data from a public storage account. Here the COPY statement's defaults match the format of the line item csv file.

```
COPY INTO dbo.[lineitem] FROM 'https://unsecureaccount.blob.core.windows.net/customerdatasets/folder1/lineitem.csv'
```

The default values for csv files of the COPY command are:

- DATEFORMAT = Session DATEFORMAT
- MAXERRORS = 0
- COMPRESSION default is uncompressed
- FIELDQUOTE = ""
- FIELDTERMINATOR = ","
- ROWTERMINATOR = '\n'
- FIRSTROW = 1
- ENCODING = 'UTF8'
- FILE_TYPE = 'CSV'
- IDENTITY_INSERT = 'OFF'

This example loads files specifying a column list with default values.

```
--Note when specifying the column list, input field numbers start from 1
COPY INTO test_1 (Col_one default 'myStringDefault' 1, Col_two default 1 3)
FROM 'https://myaccount.blob.core.windows.net/myblobcontainer/folder1/'
WITH (
    FILE_TYPE = 'CSV',
    CREDENTIAL=(IDENTITY= 'Storage Account Key', SECRET='<Your_Account_
Key>'),
    --CREDENTIAL should look something like this:
    --CREDENTIAL=(IDENTITY= 'Storage Account Key', SECRET='x6RWv4It5F2msn-
jelv3H4DA80n0PQW0daPdw43jM0nyetx4c6CpDkj3986DX5AHFMIf/YN4y6kkCnU81b+Wx0P-
j+6MDw=='),
    FIELDQUOTE = '',
    FIELDTERMINATOR=',',
    ROWTERMINATOR='0x0A',
    ENCODING = 'UTF8',
    FIRSTROW = 2
)
```

The following example loads files that use the line feed as a row terminator such as a UNIX output. This example also uses a SAS key to authenticate to Azure blob storage.

```
COPY INTO test_1
FROM 'https://myaccount.blob.core.windows.net/myblobcontainer/folder1/'
WITH (
    FILE_TYPE = 'CSV',
    CREDENTIAL=(IDENTITY= 'Shared Access Signature', SECRET='<Your_SAS_To-
ken>'),
    --CREDENTIAL should look something like this:
    --CREDENTIAL=(IDENTITY= 'Shared Access Signature',
SECRET='?sv=2018-03-28&ss=bfqt&srt=sco&sp=rl&st=2016-10-17T20%3A14%3A55Z&se
=2021-10-18T20%3A19%3A00Z&sig=IEoOdmeYnE9%2FKiJDSHFSYsz4AkNa%2F%2BTx-
61FuQ%2FFKHefqoBE%3D'),
    FIELDQUOTE = '',
    FIELDTERMINATOR=';',
    ROWTERMINATOR='0X0A',
    ENCODING = 'UTF8',
    DATEFORMAT = 'ymd',
    MAXERRORS = 10,
    ERRORFILE = '/errorsfolder',--path starting from the storage container
    IDENTITY_INSERT = 'ON'
)
```

Load a text file with non-standard row delimiters

One of the realities of data engineering, is that many times we need to process imperfect data. That is to say, data sources that contain invalid data formats, corrupted records, or custom configurations such as non-standard delimiters.

Successfully load using COPY

One of the advantages COPY has over PolyBase is that it supports custom column and row delimiters.

Suppose you have a nightly process that ingests regional sales data from a partner analytics system and saves the files in the data lake. The text files use non-standard column and row delimiters where columns are delimited by a . and rows by a ,:

```
20200421.114892.130282.159488.172105.196533,20200420.109934.108377.122039.1  
01946.100712,20200419.253714.357583.452690.553447.653921
```

The data has the following fields: Date, NorthAmerica, SouthAmerica, Europe, Africa, and Asia. They must process this data and store it in Synapse Analytics.

1. In the query window, replace the script with the following to create the DailySalesCounts table and load data using the COPY statement. As before, be sure to replace YOURACCOUNT with the name of your ADLS Gen2 account:

```
CREATE TABLE [wwi_staging].DailySalesCounts  
(  
    [Date] [int] NOT NULL,  
    [NorthAmerica] [int] NOT NULL,  
    [SouthAmerica] [int] NOT NULL,  
    [Europe] [int] NOT NULL,  
    [Africa] [int] NOT NULL,  
    [Asia] [int] NOT NULL  
)  
GO
```

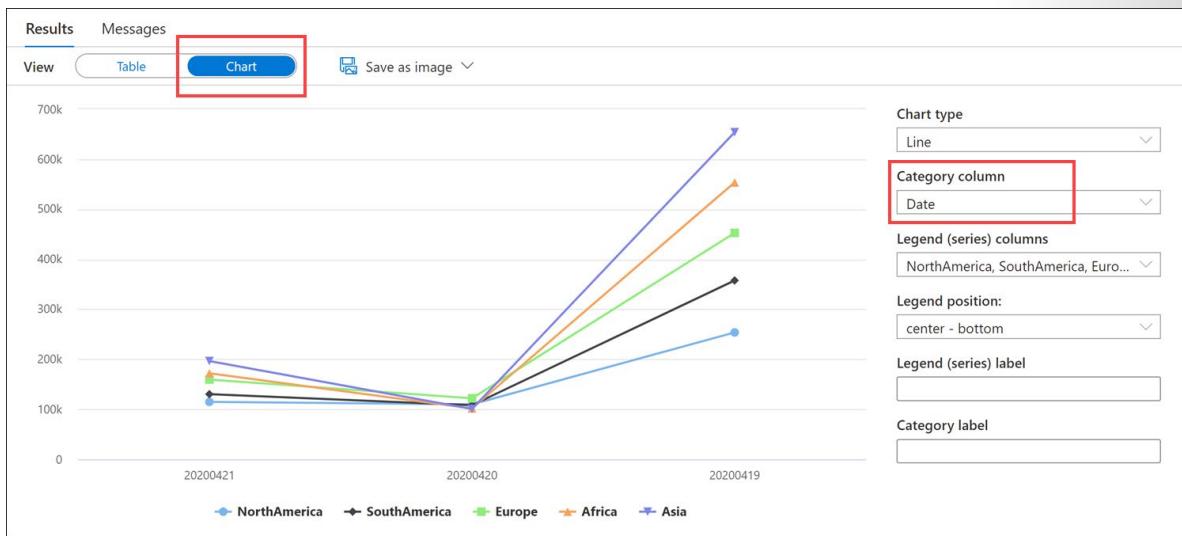
```
-- Replace <PrimaryStorage> with the workspace default storage account name.  
COPY INTO wwi_staging.DailySalesCounts  
FROM 'https://YOURACCOUNT.dfs.core.windows.net/wwi-02/campaign-analytics/dailycounts.txt'  
WITH (  
    FILE_TYPE = 'CSV',  
    FIELDTERMINATOR='.',  
    ROWTERMINATOR=''  
)  
GO
```

Notice the FIELDTERMINATOR and ROWTERMINATOR properties that help us correctly parse the file.

2. Select **Run** from the toolbar menu to execute the SQL command.
3. In the query window, replace the script with the following to view the imported data:

```
SELECT * FROM [wwi_staging].DailySalesCounts  
ORDER BY [Date] DESC
```

4. Select **Run** from the toolbar menu to execute the SQL command.
5. Try viewing the results in a Chart and set the **Category column** to Date:



Attempt to load using PolyBase

Let's try this same operation using PolyBase.

1. In the query window, replace the script with the following to create a new external file format, external table, and load data using PolyBase:

```
CREATE EXTERNAL FILE FORMAT csv_dailysales
WITH (
    FORMAT_TYPE = DELIMITEDTEXT,
    FORMAT_OPTIONS (
        FIELD_TERMINATOR = ',',
        DATE_FORMAT = '',
        USE_TYPE_DEFAULT = False
    )
);
GO
```

```
CREATE EXTERNAL TABLE [wwi_external].DailySalesCounts
(
    [Date] [int] NOT NULL,
    [NorthAmerica] [int] NOT NULL,
    [SouthAmerica] [int] NOT NULL,
    [Europe] [int] NOT NULL,
    [Africa] [int] NOT NULL,
    [Asia] [int] NOT NULL
)
WITH
(
    LOCATION = '/campaign-analytics/dailycounts.txt',
    DATA_SOURCE = ABSS,
    FILE_FORMAT = csv_dailysales
)
GO
```

```
INSERT INTO [wwi_staging].[DailySalesCounts]
SELECT *
FROM [wwi_external].[DailySalesCounts]
```

2. Select **Run** from the toolbar menu to execute the SQL command.

You should see an error similar to: Failed to execute query. Error: HdfsBridge::recordReaderFillBuffer - Unexpected error encountered filling record reader buffer: HadoopExecutionException: Too many columns in the line..

Why is this? According to **PolyBase documentation¹**:

The row delimiter in delimited-text files must be supported by Hadoop's LineRecordReader. That is, it must be either \r, \n, or \r\n. These delimiters are not user-configurable.

This is an example of where COPY's flexibility gives it an advantage over PolyBase.

Knowledge check

Question 1

How does splitting source files help maintain good performance when loading into Synapse Analytics?

- optimized processing of smaller file sizes.
- Compute node to storage segment alignment.
- Reduced possibility of data corruptions.

Question 2

Which Workload Management capability manages minimum and maximum resource allocations during peak periods?

- Workload Isolation.
- Workload Importance.
- Workload Containment.

Question 3

Which T-SQL Statement loads data directly from Azure Storage?

- LOAD DATA.
- COPY.
- INSERT FROM FILE.

Summary

In this module, you have learned the techniques to optimize performance when loading data into Azure Synapse Analytics SQL Pools.

¹ <https://docs.microsoft.com/sql/t-sql/statements/create-external-file-format-transact-sql?view=sql-server-ver15#limitations-and-restrictions>

In this module, you have:

- Understood data loading design goals
- Explained loading methods into Azure Synapse Analytics
- Managed source data files
- Managed singleton updates
- Set-up dedicated data loading accounts
- Managed concurrent access to Azure Synapse Analytics
- Implemented Workload Management
- Simplified ingestion with the Copy Activity

Petabyte-scale ingestion with Azure Data Factory

Introduction

The first stage of data integration work is to ingest source data from any number of systems into a destination, or an intermediary data store. Azure Data Factory provides differing ingestion methods that can connect to nearly a hundred different data connectors. The method you choose will largely depend on your current skillset, or preference for working with a technology to ingest data.

Regardless of which method you choose, you must set up the appropriate infrastructure to support the data ingestion method through integration runtimes. In addition, you must also make sure that the data ingestion is secure in each data store, and whilst in transit

Learning objectives

In this module, you will:

- Describe the data ingestion methods
- Understand the connectors that are available
- Use the Copy Activity
- Manage a self-hosted integration runtime
- Setup and Azure integration runtime
- Describe data ingestion security considerations

List the data factory ingestion methods

Azure Data Factory can accommodate organizations that are embarking on data integration projects from the differing starting point. It is rare for a data migration project to be a green field project. Typically, many data integration workflows must consider existing pipelines that have been created on previous projects, with different dependencies and using different technologies. To that end, there are various ingestion methods that can be used to extract data from a variety of sources.

Ingesting data using the Copy Activity

Use this method to build code-free data ingestion pipelines that don't require any transformation during the extraction of the data. The Copy Activity has support for over 100 native connectors. This method can suit green field projects that have a simple method of extraction to an intermediary data store. An example of ingesting data using the Copy Activity can include extracting data from multiple source database systems and outputting the data to files in a data lake store. The benefit of this ingestion method is that they are simple to create, but they are not able to deal with sophisticated transformations or business logic.

Ingesting data using compute resources

Azure Data Factory can call on compute resources to process data by a data platform service that may be better suited to the job. A great example of this is that Azure Data Factory can create a pipeline to an analytical data platform such as Spark pools in an Azure Synapse Analytics instance to perform a complex

calculation, which generates new data. This data is then ingested back into the pipeline for further downstream processing. There a wide range of computing resource, and the associated activities that they can perform as shown in the following table:

Compute environment	activities
On-demand HDInsight cluster or your own HDInsight cluster	Hive, Pig, Spark, MapReduce, Hadoop Streaming
Azure Batch	Custom activities
Azure Machine Learning Studio Machine	Learning activities: Batch Execution and Update Resource
Azure Machine Learning	Azure Machine Learning Execute Pipeline
Azure Data Lake Analytics	Data Lake Analytics U-SQL
Azure SQL, Azure SQL Data Warehouse, SQL Server	Stored Procedure
Azure Databricks	Notebook, Jar, Python
Azure Function	Azure Function activity

Ingesting data using SSIS packages

Many organizations have decades of development investment in SQL Server Integration Services (SSIS) packages that contain both ingestion and transformation logic from on-premises and cloud data stores. Azure Data Factory provides the ability to lift and shift existing SSIS workload, by creating an Azure-SSIS Integration Runtime to natively execute SSIS packages, and will enable you to deploy and manage your existing SSIS packages with little to no change using familiar tools such as SQL Server Data Tools (SSDT) and SQL Server Management Studio (SSMS), just like using SSIS on-premises.

Describe data factory connectors

Connectors are Azure Data Factory objects that enable your Linked Services and Datasets to connect to a wide variety of data sources and sinks. These can include connections to Azure resources and third-party connectors such as Amazon S3 or Google cloud. There are nearly 100 connectors that are available, and they work with the Copy, Data Flow, Look up, Get Metadata, and Delete activities that can be found within Azure Data Factory.

The file formats that are supported include:

- Avro format
- Binary format
- Delimited text format
- JSON format
- ORC format
- Parquet format

There are too many data stores to list, but the following table lists the categories of data stores and two examples of the types of connectors that exist

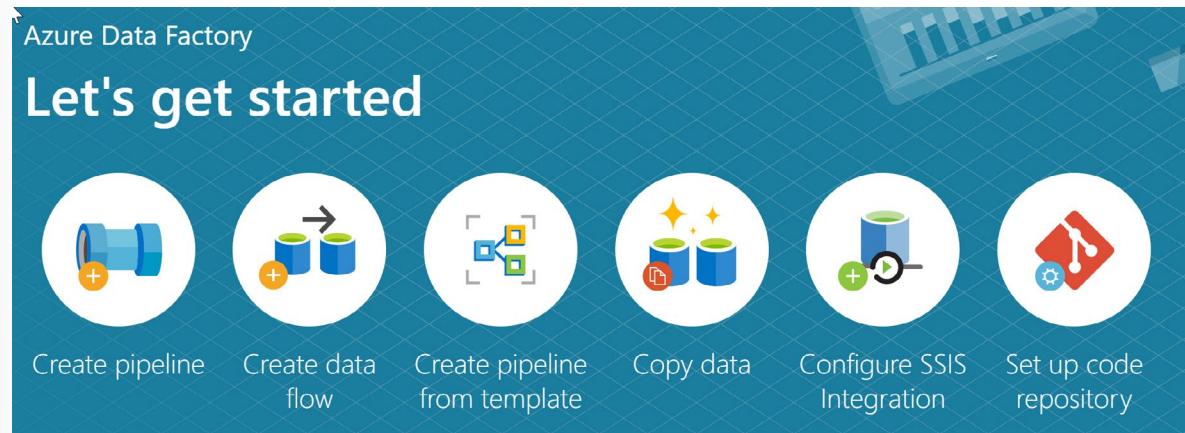
Category	Data Store example
Azure	Azure Data Lake Store, Azure Synapse Analytics
Databases	Netezza, Greenplum
NoSQL stores	Cassandra, MongoDB

Category	Data Store example
File	FTP, Google Cloud Storage
Generic protocols	REST, ODBC
Services & Apps	Dynamics, SalesForce

The list of connectors is constantly evolving. You can keep upto date with the latest list, and the activity support by looking at the **connectors overview page**²

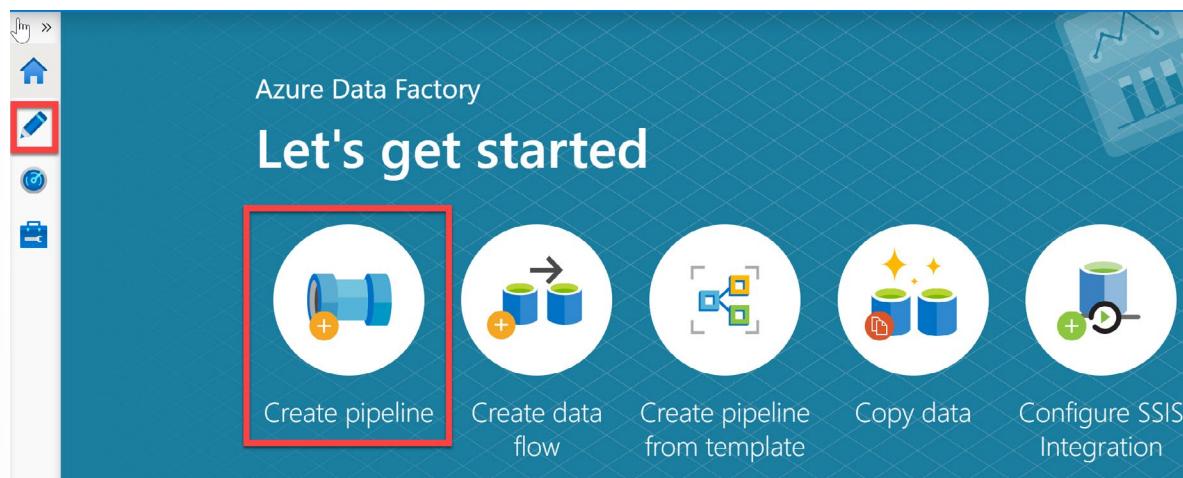
Use the data factory copy activity

Once the creation of the Data Factory instance is complete, you can go to the resource where you can begin to create your data pipelines by clicking on the **Author & Monitor** button. This will open up the following screen:



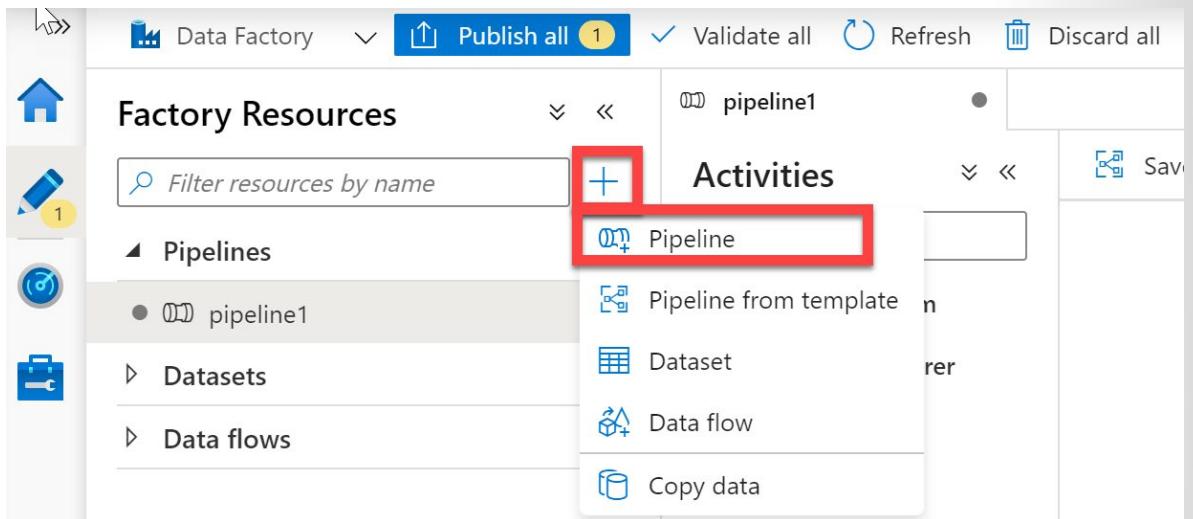
The first step in your pipeline is creating a Copy Activity that copies data between the source and destination using the following steps.

1. Open the **authoring canvas** by clicking on the **pencil icon** on the left sidebar or the create pipeline button to open the authoring canvas.

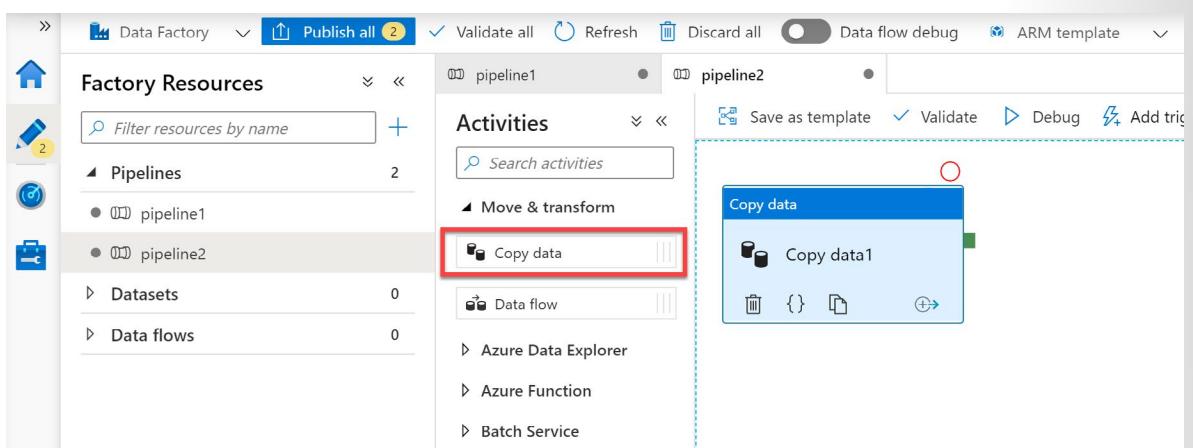


2. Create the **pipeline**. Click on the + button in the Factory Resources pane and select **Pipeline**.

² <https://docs.microsoft.com/azure/data-factory/connector-overview>

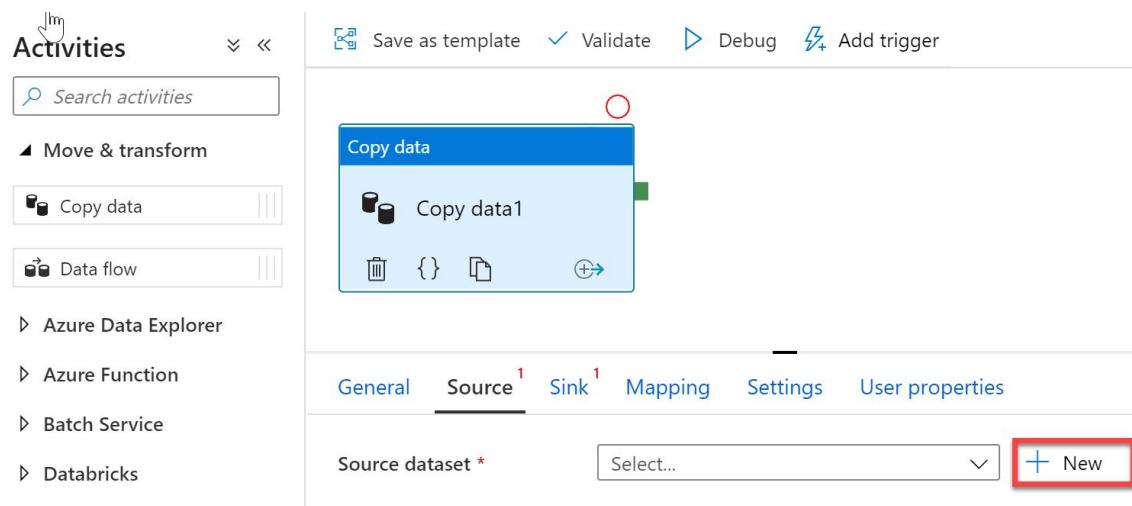


3. Add a **copy activity**. In the **Activities** pane, open the **Move and Transform** accordion and drag the **Copy Data** activity onto the pipeline canvas.



With the Copy Activity added, you then start to define the source data

1. In the **Source** tab of the Copy Activity settings, click **+ New** to select a data source.



2. For example, In the data store list, select the **Amazon S3** tile and click **continue**

New 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

The screenshot shows a search interface for selecting a data store. The search bar contains 'Ama'. Below it, a list of categories is shown: All, Azure, Database, File, Generic protocol, NoSQL, Services and apps. The 'All' category is selected. Three data store tiles are displayed: Amazon Marketplace Web Service (with a shopping cart icon), Amazon Redshift (with a cylinder icon), and Amazon S3 (with a bucket icon). The Amazon S3 tile is highlighted with a blue dashed border.

3. In the **file format list**, select the **DelimitedText** format tile and click **continue**

Select format

Choose the format type of your data



Avro



Binary



DelimitedText



Excel



Json



ORC



Parquet



XML

4. In **Set Properties** window, give your dataset an understandable **name** and click on the **Linked Service dropdown**. If you have not created your S3 Linked Service, select **New**.

Set properties

Name

MoviesS3

Linked service *

Select...

Filter...

Select...

+ New



5. Specific to the S3 Linked Service configuration pane, specify your S3 **access key** and **secret key**. The Data Factory service encrypts credentials with certificates managed by Microsoft. For more information, see Data Movement Security Considerations. To verify your credentials are valid, click **Test Connection**. Click **Create** when finished.

New Linked Service (Amazon S3) X

Name *
S3

Description

Connect via integration runtime * i
AutoResolveIntegrationRuntime ▼

Access Key ID *

Secret Access Key	Azure Key Vault
Secret Access Key * 	

Service URL i

Annotations

+ New

► Advanced i

Create Test connection Cancel

6. Once you have created and selected the linked service, specify the rest of your dataset settings. These settings specify how and where in your connection you want to pull the data. Click **Finish** once completed.

Set properties X

Name
MoviesS3

Linked service *
AmazonS31 ▼

[Edit connection](#)

File path
daperlov-test / Directory / moviesDB.csv [Browse](#) ▾

First row as header

Import schema
 From connection/store From sample file None

OK Next->Advanced Back Cancel

7. To verify your dataset is configured correctly, click **Preview Data** in the Source tab of the Copy Activity to get a small snapshot of your data.

movie	title	genres	year	Rating	Rotten Tomato
108583	Fawlty Towers (1975)	Comedy	1975	1	54
32898	Trip to the Moon, A (Voyage dans la lune, Le)	Action Adventure Fantasy Sci-Fi	1902	7	80
7065	Birth of a Nation, The	Drama War	1915	6	92
7243	Intolerance: Love's Struggle Throughout the Ages	Drama	1915	4	82
62383	20,000 Leagues Under the Sea	Action Adventure Sci-Fi	1915	9	92
8511	Immigrant, The	Comedy	1917	4	59
3309	Dog's Life, A	Comedy	1917	3	83
72626	Billy Blazes, Esq.	Comedy Western	1918	2	63
6987	Cabinet of Dr. Caligari, The (Cabinet des Dr. Caligari, Das)	Crime Fantasy Horror	1919	4	63

With the source data defined, then you will define the sink into which the data will be loaded. In this example the sink will be Azure Data Lake Storage Gen2 by performing the following steps:

1. In the **Sink** tab, click **+ New**

The screenshot shows the 'Copy data' dialog with a dataset named 'Copy data1'. Below the list are icons for delete, copy, and new. The 'Sink' tab is selected. A red box highlights the '+ New' button in the bottom right corner of the sink configuration area.

2. Select the **Azure Data lake Storage Gen2** tile and click **continue**

New Dataset

Select a Data Store

Search

All Azure Database File Generic protocol Services and apps

Azure Blob Storage	Azure Cosmos DB (MongoDB API)	Azure Cosmos DB (SQL API)
Azure Data Explorer (Kusto)	Azure Data Lake Storage Gen1	Azure Data Lake Storage Gen2

3. In **Set Properties** side nav, give your dataset an understandable **name** and click on the **Linked Service dropdown**. If you have not created your ADLS Linked Service, select **New**.

Set properties

Name
MoviesS3

Linked service *

Select...
Filter...
Select...
+ New

4. In the ADLS linked service configuration pane, select your **authentication method** and **enter your credentials**. In the example below, an account key and selected my storage account from the drop down.

New Linked Service (Azure Data Lake Storage Gen2) X

Name *
ADLS

Description

Connect via integration runtime *
AutoResolveIntegrationRuntime

Authentication method
Account key

Account selection method
 From Azure subscription Enter manually

Azure subscription
ADFCustomerDemos (222f1459-6ebd-4896-82ab-652d5f6883cf)

Storage account name *
adfdemo

Annotations
+ New

► Advanced i

5. Once you have configured your linked service, enter in the **ADLS dataset configuration**. Click **finish** once completed.

Set properties

Name
MoviesADLS

Linked service *
AzureDataLakeStorage1

[Edit connection](#)

File path
sample-data / output/ / File [Browse](#)

First row as header

Import schema
 From connection/store From sample file None

Select file
moviesDB.csv [Browse](#)

At this point, you have fully configured your copy activity.

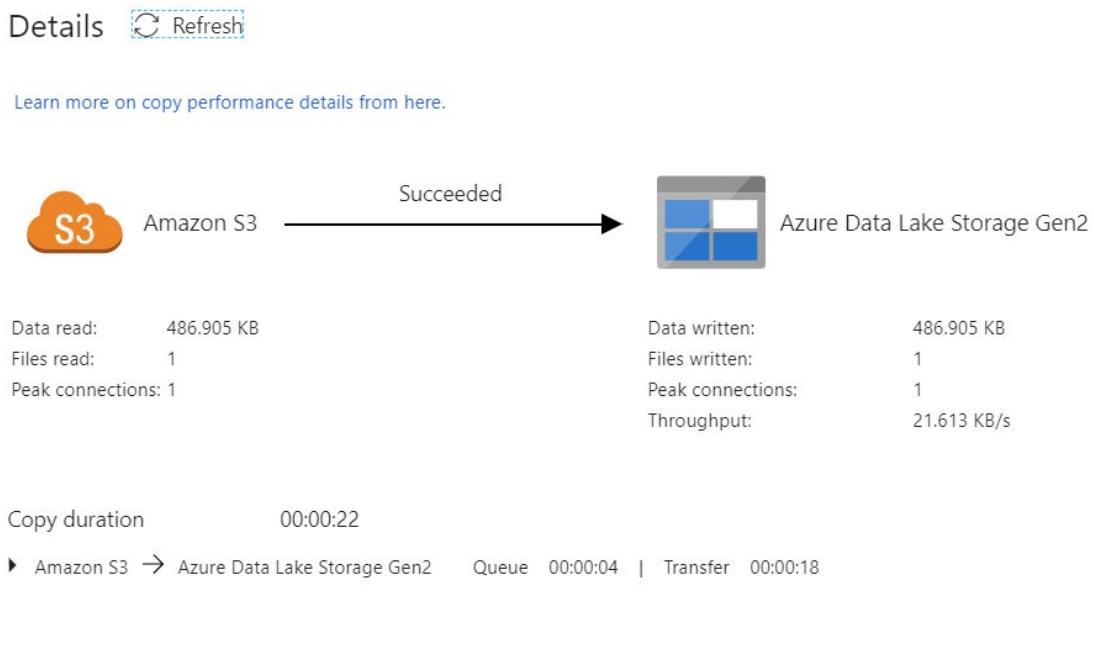
1. To test it out, click on the **Debug** button at the top of the pipeline canvas. This will start a pipeline debug run.



2. To monitor the progress of a pipeline debug run, click on the Output tab of the pipeline

General	Parameters	Variables	Output
Pipeline Run ID: ee9431fe-c67c-420d-9b9e-2ece8fd990f1			
NAME	TYPE	RUN START	DURATION
MoveFromS3ToA... Copy		07/10/2019 3:26 PM	00:00:24
STATUS	ACTIONS	RUNID	
✓ Succeeded	↻ ⟳ 🔗	18cb6b00-0407-48df-a928-19e7073449df	

3. To view a more detailed description of the activity output, click on the eyeglasses icon. This will open up the copy monitoring screen which provides useful metrics such as Data read/written, throughput and in-depth duration statistics.



To verify the copy worked as expected, open up your ADLS gen2 storage account and check to see your file was written as expected

Manage the self-hosted integration runtime

In Data Factory, an activity defines the action to be performed. A linked service defines a target data store or a compute service. An integration runtime provides the bridge between the activity and linked services.

Self-hosted integration runtime

A self-hosted integration runtime is capable of:

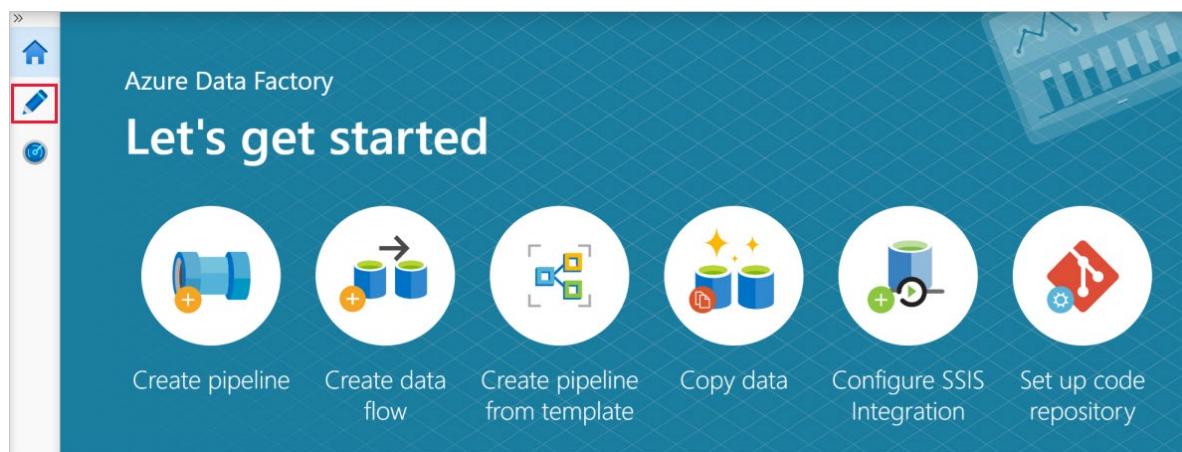
- Running copy activity between a cloud data stores and a data store in private network.
- Dispatching the following transform activities against compute resources in on-premises or Azure Virtual Network:
 - HDInsight Hive activity (BYOC-Bring Your Own Cluster)
 - HDInsight Pig activity (BYOC)
 - HDInsight MapReduce activity (BYOC)
 - HDInsight Spark activity (BYOC)
 - HDInsight Streaming activity (BYOC)
 - Machine Learning Batch Execution activity
 - Machine Learning Update Resource activities

- Stored Procedure activity
- Data Lake Analytics U-SQL activity
- Custom activity (runs on Azure Batch)
- Lookup activity
- Get Metadata activity.

The self-hosted integration runtime is logically registered to the Azure Data Factory and the compute resource used to support its functionality as provided by you. Therefore there is no explicit location property for self-hosted IR. When used to perform data movement, the self-hosted IR extracts data from the source and writes into the destination.

Create a self-hosted Integration Runtime within Azure Data Factory

1. On the Let's get started page of Azure Data Factory UI, select the Author tab on the leftmost pane.



2. Select Manage in the leftmost pane, and select Integration runtimes. Select +New.

NAME ↑	TYPE ↑	SUB-TYPE ↑
AutoResolveIntegratio...	Azure	Public

3. On the Integration runtime setup page, select Azure, Self-Hosted, and then select Continue.

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](#)



Azure, Self-Hosted

Perform data flows, data movement and dispatch activities to external compute.



Azure-SSIS

Lift-and-shift existing SSIS packages to execute in Azure.

4. On the Integration runtime setup page, type in a name of MySelfHostedIR, 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.

Name *

(i)

MySelfHostedIR

Description

Enter description here...

Type

Self-Hosted

5. Copy and paste the authentication key. Select Download and install integration runtime.

Integration runtime setup

Settings Nodes Auto update Sharing

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

Name



MySelfHostedIR

Option 1: Express setup

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

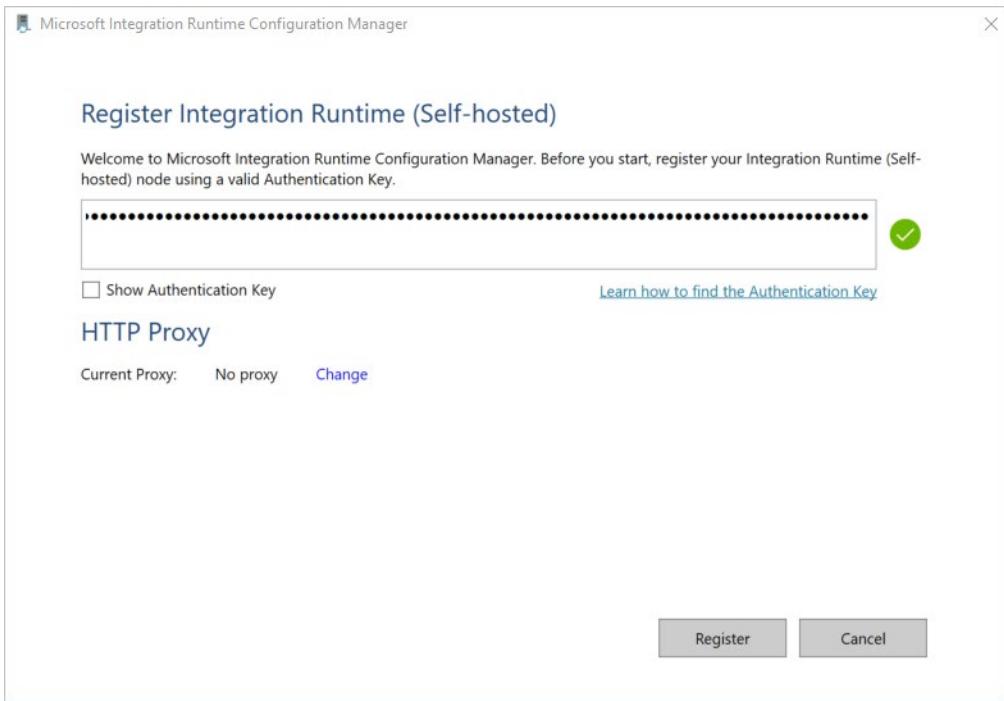
Option 2: Manual setup

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

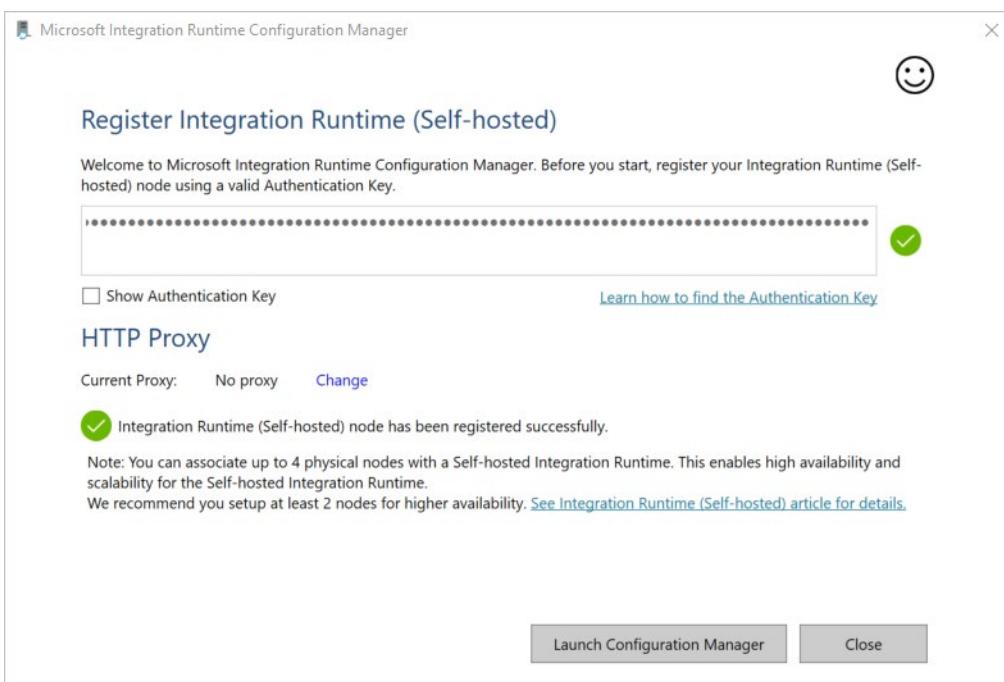
Step 2: Use this key to register your integration runtime

Name	Authentication key	Actions
Key1	IR@'	
Key2	IR@!'	

6. Download the self-hosted integration runtime on a local Windows machine. Run the installer.
7. On the Register Integration Runtime (Self-hosted) page, paste the key you saved earlier, and select Register.



8. On the New Integration Runtime (Self-hosted) Node page, select Finish.
9. After the self-hosted integration runtime is registered successfully, you see the following window:



Automated deployments

You can also set up a self-hosted IR on an Azure VM via an Azure Resource Manager template, or by using PowerShell

1. Run the following command in PowerShell

```
Set-AzDataFactoryV2IntegrationRuntime -ResourceGroupName $resourceGroupName -DataFactoryName $dataFactoryName -Name $selfHostedIntegrationRuntimeName -Type SelfHosted -Description "selfhosted IR description"
```

2. Download and install the self-hosted integration runtime on a local machine.
3. Retrieve the authentication key and register the self-hosted integration runtime with the key. Here is a PowerShell example:

```
Get-AzDataFactoryV2IntegrationRuntimeKey -ResourceGroupName $resourceGroupName -DataFactoryName $dataFactoryName -Name $selfHostedIntegrationRuntimeName
```

Set up the Azure integration runtime

In Data Factory, an activity defines the action to be performed. A linked service defines a target data store or a compute service. An integration runtime provides the bridge between the activity and linked services.

Azure integration runtime

An Azure integration runtime is capable of:

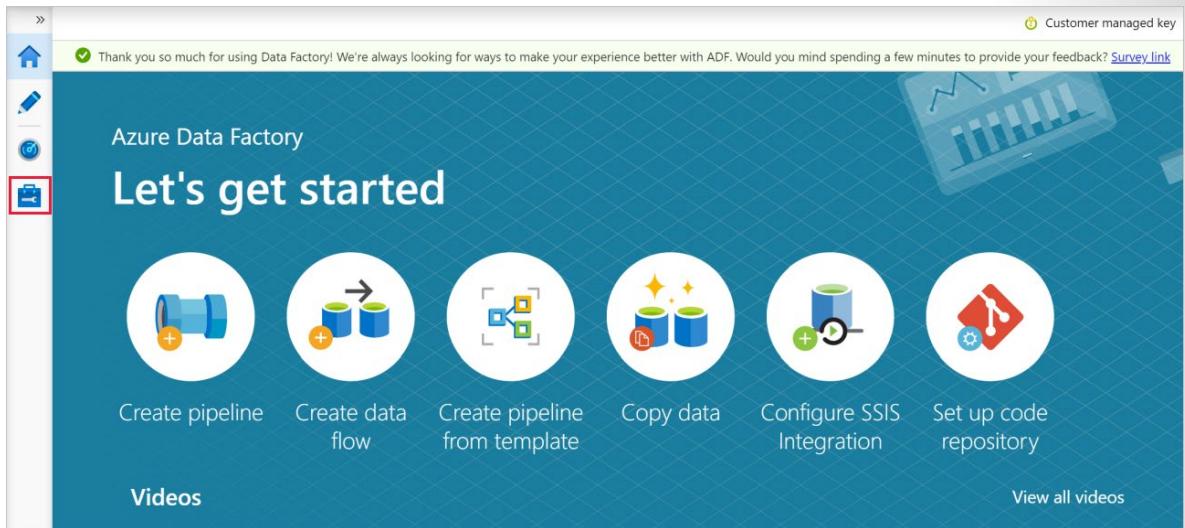
- Running Data Flows in **Azure**
- Running Copy Activity **between cloud data stores**
- Dispatching the following transform activities in **public network**: Databricks Notebook/ Jar/ Python activity, HDInsight Hive activity, HDInsight Pig activity, HDInsight MapReduce activity, HDInsight Spark activity, HDInsight Streaming activity, Machine Learning Batch Execution activity, Machine Learning Update Resource activities, Stored Procedure activity, Data Lake Analytics U-SQL activity, .NET custom activity, Web activity, Lookup activity, and Get Metadata activity.

You can set a certain location of an Azure IR, in which case the data movement or activity dispatch will happen in that specific region. If you choose to use the auto-resolve Azure IR which is the default, ADF will make a best effort to automatically detect your sink and source data store to choose the best location either in the same region if available or the closest one in the same geography for the Copy Activity. For anything else, it will use the IR in the Data Factory region. Azure Integration Runtime also has support for virtual networks.

Create and configure Azure integration runtime

Use the following steps to create an Azure IR using Azure Data Factory UI.

1. On the Let's get started page of Azure Data Factory UI, select the Manage tab from the leftmost pane.



2. Select Integration runtimes on the left pane, and then select +New.

The screenshot shows the 'Integration runtimes' page in the Azure Data Factory interface. The left sidebar has a red box around the 'Integration runtimes' item. The main panel title is 'Integration runtimes' with a sub-instruction: 'The integration runtime (IR) is the compute infrastructure to provide the following'. It includes a 'Learn more' link, a '+ New' button (which is highlighted with a red box), and a 'Refresh' button. Below this, it says 'Showing 1 - 1 of 1 items' and lists one item: 'AutoResolveIntegratio...' with 'Azure' under 'TYPE' and 'Public' under 'SUB-TYPE'. The table has columns for NAME, TYPE, and SUB-TYPE.

3. On the Integration runtime setup page, select Azure, Self-Hosted, and then select Continue.
4. On the following page, select Azure to create an Azure IR, and then select Continue.

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-premise / 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](#)

Buttons:

Continue **Back** **Cancel**

5. Enter a name for your Azure IR, and select Create.

Integration runtime setup

The Data Factory manages the integration runtime in Azure to connect to required data source/destination or external compute in public network. The compute resource is elastic allocated based on performance requirement of activities.

Name *

(i)

Description

Type

Virtual network configuration (Preview)

(i)

Disable Enable

i This data factory is not yet enabled with Virtual Network feature. [Request here to opt-in](#)

Region *

▼

▲ Data flow run time

Compute type *

▼

Core count *

▼

Time to live

▼

Billing for data flows is based upon the type of compute you select and the number of cores selected per hour. If you set a TTL, then the minimum billing time will be that amount of time. Otherwise, the time billed will be based on the execution time of your data flows and the time of your debug sessions. Note that debug sessions will incur a minimum of 60 minutes of

You'll see a pop-up notification when the creation completes. On the Integration runtimes page, make sure that you see the newly created IR in the list.

Automated deployments

You can also set up Azure IR via an Azure Resource Manager template, or by using PowerShell

1. Run the following command in PowerShell

```
Set-AzDataFactoryV2IntegrationRuntime -DataFactoryName "SampleV2DataFactory1" -Name "MySampleAzureIR" -ResourceGroupName "ADFV2SampleRG" -Type Managed -Location "West Europe"
```

When Azure IR, the type must be set to Managed. You do not need to specify compute details because it is fully managed elastically in cloud.

Understand data ingestion security considerations

Data integration requires the secure handling of data that is both at rest and in transit. Prior to the creation of a data integration solution, the design stage must consider the security requirements. This will initially provide coverage of the source systems and how the data from them are ingested. However, there is a need for a holistic approach, that also encapsulates the security of intermediary data stores in the following areas

Network

There are a number of issues that you should be aware of pertaining to setting up network security. You may have to work with the Azure Administrator in your organization to manage some of these areas.

Use virtual networks to secure Azure resources

Virtual networks allow secure communication between Azure services, or with servers that exist in on-premises network. Virtual networks or VNets are the fundamental building block for configuring private networks. It enables the secure communication between Azure resources, services on the internet, and with server on on-premises networks.

Azure Data Factory may ingest data from an on-premises server, or a virtual machine that is hosted within Azure. To achieve this, a self-hosted integration runtime can be deployed on a server inside a virtual network. To restrict access, you should configure a Network Security Group (NSG) to only allow administrative access.

When using Azure-SSIS Integration runtime, you are given the option to join a virtual network. Accepting this option enables Azure Data Factory to create network resources, an example of which is that a Network Security Group is automatically created by Azure Data Factory, and port 3389 is open to all traffic by default. Lock this down to ensure that only your administrators have access.

Use services to detect and prevent intrusions

You can deny communication with known IP addresses by enabling the distributed denial of service (DDoS) protection standard on the virtual networks on which the integration runtime is hosted. In addition, you can use the Azure Security Center Integrated Threat Intelligence to deny communications with known malicious or unused Internet IP addresses.

Azure Firewall with Threat Intelligence can be used to control network access. If intrusion detection and/or prevention based on payload inspection is required, you can redirect traffic to a firewall appliance via Azure ExpressRoute force tunneling or to a Network Virtual Appliance that supports this capability

Simplify the management of security rules using network service tags

Virtual networks can be configured with service tags. Service tags enable you to group together IP address prefixes from a given Azure service for administrative purposes. Using Service tags, you can create network security rules in Network Security Groups based on service tags to reduce the administrative overheads. By specifying the service tag name (e.g., DataFactoryManagement) in the appropriate source or destination field of a rule, you can allow or deny inbound traffic for the corresponding service.

Identity and access control

Managing access to your data is an important area to consider. Here are some areas to be aware of.

Administrative accounts

The administrative accounts that are used to work and manage Azure Data Factory should be dedicated, known accounts that are monitored and managed on a regular basis to ensure they are not compromised. To create Data Factory instances, the user account that you use to sign in to Azure must be a member of the contributor or owner role, or an administrator of the Azure subscription. For high security environments, consider using dedicated machines for administrative access for any ADF administrative tasks

Use Active Directory to make use of single sign-on

Register service principals within Azure Active Directory to take advantage of token management so that your Azure Data Factory service streamline its authentication across Azure resources. A data factory can be associated with a managed identity for Azure resources that represents the specific data factory. You can use this managed identity for Azure SQL Database authentication. The designated factory can access and copy data from or to your database by using this identity.

Data protection

You may have to place special considerations for specific types of data, such as medical data, or financial data in the following areas.

Use Role Based Access Control (RBAC) to control access to the resources

Use RBAC on the data sources to control access to the data to Azure Data Factory service principal.

Sensitive data

There are a number of considerations that you should account for when working with sensitive data, including:

- Maintaining a list of the data stores that contain sensitive information
- Isolate the systems that store or process sensitive information
- Monitor and block unauthorized transfer of sensitive information
- Encrypt any sensitive information that goes in transit

- Encrypt any sensitive information in rest

Logging and monitoring

It is also just as important to understand who is accessing your data, and your security considerations should involve the following areas.

Configure central security log management

Use Azure Monitor to centralize the storage of ingestion logs that are generated by Azure Data Factory, and query them using Log Analytics. In addition, set up a strategy to store the logs long term in Azure Storage Accounts so that you have the data to establish baselines for ADF ingestion activities

Monitor and log the configuration and network packet traffic of virtual networks, subnets, and NICs

Enable network security group (NSG) flow logs for the NSG protecting your Integration Runtime deployment and send logs into an Azure Storage Account for traffic auditing. You may also send NSG flow logs to a Log Analytics workspace and use Traffic Analytics to provide insights into traffic flow in your Azure cloud.

Enable audit logging

You can use Azure Data Factory diagnostic settings to configure diagnostic logs to track pipeline-run data, which is retained for 45 days. You can save the diagnostic logs to Azure Storage accounts for future analysis.

Enable alerts on activities

Diagnostic setting for Azure Data Factory that send logs to Log Analytics can have alerts configured for a set of pre-defined set of conditions that can alert an administrator to activities

Follow standard logging and monitoring standard within your organization

Check your organizations standards for logging and monitoring and snap to the standard including:

- Audit logging
- Security logs
- Anti-malware logging
- Log retention policies

Knowledge check

Question 1

In Azure Data Factory authoring tool, where would you find the Copy data activity?

- Move & Transform
- Batch Service
- Databricks

Question 2

You want to ingest data from a SQL Server database hosted on an on-premises Windows Server. What integration runtime is required for Azure Data Factory to ingest data from the on-premises server?

- Azure-SSIS Integration Runtime
- Self-Hosted Integration Runtime
- Azure Integration Runtime

Question 3

By default, how long are the Azure Data Factory diagnostic logs retained for?

- 15 days
- 30 days
- 45 days

Summary

Data Ingestion can operate at petabyte scale using an ingestion method that is suited to your skillset. Using the Copy Activity method provides the simplest method to set up your ingestion mechanism. Regardless of the method that you use, you should be aware that for data transfers from on-premises servers, that a self-hosted integration runtime is required to allow this. For SSIS package execution, the Azure-SSIS integration runtime is more appropriate. It's also important to ensure that the data is ingested securely.

Important: Should you be working in your own subscription while running through this content, it is best practice at the end of a project to identify whether or not you still need the resources you created. Resources left running can cost you money. You can delete resources one by one, or just delete the resource group to get rid of the entire set.

Module Lab information

Lab 5 - Ingest and load data into the data warehouses

- **Estimated Time:** 50 - 60 minutes
- **Lab files:** The files for this lab are located in the *Instructions\Labs\07* folder.

Lab overview

This lab teaches students how to ingest data into the data warehouse through T-SQL scripts and Synapse Analytics integration pipelines. The student will learn how to load data into Synapse dedicated SQL pools with PolyBase and COPY using T-SQL. The student will also learn how to use workload management along with a Copy activity in a Azure Synapse pipeline for petabyte-scale data ingestion

Lab objectives

After completing this lab, you will be able to:

- Perform petabyte-scale ingestion with Azure Synapse Pipelines
- Import data with PolyBase and COPY using T-SQL
- Use data loading best practices in Azure Synapse Analytics

Module summary

module-summary

In this module, you have learned how to use the best practices you need to adopt to load data into a data warehouse in Azure Synapse Analytics. You understand the various methods that can be used to ingest data between various data stores using Azure Data Factory, and have performed petabyte-scale ingestion with Azure Data Factory.

Learning objectives

In this module, you have learned about:

- Data loading best practices in Azure Synapse Analytics
- Petabyte-scale ingestion with Azure Data Factory

Post Course Review

After the course, consider reading **Best practices for loading data using dedicated SQL pools in Azure Synapse Analytics³** for more guidance on data ingestion and loading.

Important

Remember

Should you be working in your own subscription while running through this content, it is best practice at the end of a project to identify whether or not you still need the resources you created. Resources left running can cost you money.

You can delete resources one by one, or just delete the resource group to get rid of the entire set.

³ <https://docs.microsoft.com/en-us/azure/synapse-analytics/sql-data-warehouse/guidance-for-loading-data>

Answers

Question 1

How does splitting source files help maintain good performance when loading into Synapse Analytics?

- optimized processing of smaller file sizes.
- Compute node to storage segment alignment.
- Reduced possibility of data corruptions.

Explanation

Correct. SQL Pools have 60 storage segments. Compute can also scale to 60 nodes and so optimizing for alignment of these 2 resources can dramatically decrease load times.

Question 2

Which Workload Management capability manages minimum and maximum resource allocations during peak periods?

- Workload Isolation.
- Workload Importance.
- Workload Containment.

Explanation

Correct. Workload Isolation assigns maximum and minimum usage values for varying resources under load. These adjustments can be done live without having to take the SQL Pool offline.

Question 3

Which T-SQL Statement loads data directly from Azure Storage?

- LOAD DATA.
- COPY.
- INSERT FROM FILE.

Explanation

Correct. The T-SQL COPY Statement reads data from Azure Blob Storage or the Azure Data Lake and inserts it into a table within the SQL Pool.

Question 1

In Azure Data Factory authoring tool, where would you find the Copy data activity?

- Move & Transform
- Batch Service
- Databricks

Explanation

Correct. The Move & Transform section contains activities that are specific to Azure Data Factory copying data and defining data flows.

Question 2

You want to ingest data from a SQL Server database hosted on an on-premises Windows Server. What integration runtime is required for Azure Data Factory to ingest data from the on-premises server?

- Azure-SSIS Integration Runtime
- Self-Hosted Integration Runtime
- Azure Integration Runtime

Explanation

Correct. A self-hosted integration runtime can run copy activities between a cloud data store and a data store in a private network. It also can dispatch transform activities against compute resources in an on-premises network or an Azure virtual network.

Question 3

By default, how long are the Azure Data Factory diagnostic logs retained for?

- 15 days
- 30 days
- 45 days

Explanation

Correct. The Azure Data Factory diagnostic logs are retained for 45 days.

Module 6 Transform Data with Azure Data Factory or Azure Synapse Pipelines

Module introduction

module-introduction

In this module, you will learn how to perform data integration with Azure Data Factory or Azure Synapse pipelines. Students will be taught how to execute code-free transformation at scale with Azure Data Factory and Azure Synapse pipelines.

Learning objectives

In this module, you will:

- Perform Data integration with Azure Data Factory or Azure Synapse Pipelines
- Perform Code-free transformation at scale with Azure Data Factory or Azure Synapse Pipelines

Data integration with Azure Data Factory

Introduction

With the wide range of data stores available in Azure, there is the need to manage and orchestrate the movement data between them. In fact, you may want to automate a regular process of data movement as part of a wider enterprise analytical solution. Both Azure Data Factory and Azure Synapse Pipelines meet that need, and in this section, you will be introduced to this technology, its component parts, the Azure Data Factory process, and the security required to provision and manage the service.

Learning objectives

In this module, you will:

- Understand Azure Data Factory
- Describe data integration patterns
- List the Data Factory Process
- Describe the Azure Data Factory components
- Manage Azure Data Factory Security
- Setup Azure Data Factory
- Create Linked Services
- Create Datasets
- Create Activities and Pipelines
- Manage Integration runtime

Understand Azure Data Factory

The need to trigger the batch movement of data, or to set up a regular schedule is a requirement for most analytics solutions. Azure Data Factory (ADF) is the service that can be used to fulfill such a requirement. ADF provides a cloud-based data integration service that orchestrates the movement and transformation of data between various data stores and compute resources.

Azure Data Factory is the cloud-based ETL and data integration service that allows you to create data-driven workflows for orchestrating data movement and transforming data at scale. Using Azure Data Factory, you can create and schedule data-driven workflows (called pipelines) that can ingest data from disparate data stores. You can build complex ETL processes that transform data visually with data flows or by using compute services such as Azure HDInsight Hadoop, Azure Databricks, and Azure Synapse Analytics.

Much of the functionality of Azure Data Factory appears in Azure Synapse Analytics as a feature referred to as Pipelines, which enables you to integrate data pipelines between SQL Pools, Spark Pools and SQL Serverless, providing a one stop shop for all your analytical needs.

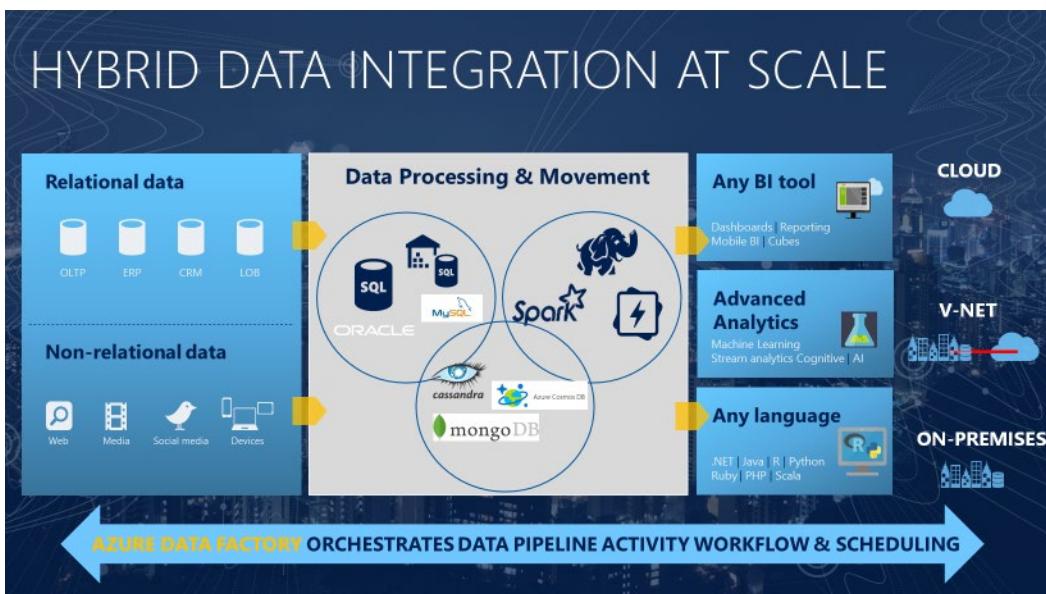
What is meant by orchestration

To use an analogy, think about a symphony orchestra. The central member of the orchestra is the conductor. The conductor does not play the instruments, they simply lead the symphony members through the entire piece of music that they perform. The musicians use their own skills to produce particular

sounds at various stages of the symphony, so they may only learn certain parts of the music. The conductor orchestrates the entire piece of music, and therefore is aware of the entire score that is being performed. They will also use specific arm movements that provide instructions to the musicians how a piece of music should be played.

ADF can use a similar approach, whilst it has native functionality to ingest and transform data, sometimes it will instruct another service to perform the actual work required on its behalf, such as a Databricks to execute a transformation query. So, in this case, it would be Databricks that performs the work, not ADF. ADF merely orchestrates the execution of the query, and then provides the pipelines to move the data onto the next step or destination.

It also provides rich visualizations to display the lineage and dependencies between your data pipelines, and monitor all your data pipelines from a single unified view to easily pinpoint issues and setup monitoring alerts.



Describe data integration patterns

Microsoft Azure provides a variety of data platform services that enables you to perform different types of analytics. Whether it is a descriptive analytics solution in a data warehouse, through to predictive analytics within HDInsight, Azure Databricks or Machine Learning Services. There is a need for a service to deal with the important aspect of data integration.

Data integration firstly involves the collection of data from one or more sources. Optionally, it typically then includes a process where the data may be cleansed and transformed, or perhaps augmented with additional data and prepared. Finally, the amalgamated data is stored in a data platform service that handles the type of analytics that you want to perform. This process can be automated by Azure Data Factory in a pattern known as Extract, Transform and Load (ETL).

Extract

During the extraction process, data engineers define the data and its source:

- **Define the data source:** Identify source details such as the resource group, subscription, and identity information such as a key or secret.

- **Define the data:** Identify the data to be extracted. Define data by using a database query, a set of files, or an Azure Blob storage name for blob storage.

Transform

- **Define the data transformation:** Data transformation operations can include splitting, combining, deriving, adding, removing, or pivoting columns. Map fields between the data source and the data destination. You might also need to aggregate or merge data.

Load

- **Define the destination:** During a load, many Azure destinations can accept data formatted as a JavaScript Object Notation (JSON), file, or blob. You might need to write code to interact with application APIs.

Azure Data Factory offers built-in support for Azure Functions. You'll also find support for many programming languages, including Node.js, .NET, Python, and Java. Although Extensible Markup Language (XML) was common in the past, most systems have migrated to JSON because of its flexibility as a semistructured data type.

- **Start the job:** Test the ETL job in a development or test environment. Then migrate the job to a production environment to load the production system.
- **Monitor the job:** ETL operations can involve many complex processes. Set up a proactive and reactive monitoring system to provide information when things go wrong. Set up logging according to the technology that will use it.

ETL tools

As a data engineer, there are several available tools for ETL. Azure Data Factory provides nearly 100 enterprise connectors and robust resources for both code-free and code-based users to accomplish their data movement and transformation needs.

Evolution from ETL

Azure has opened the way for technologies that can handle unstructured data at an unlimited scale. This change has shifted the paradigm for loading and transforming data from ETL to extract, load, and transform (ELT).

The benefit of ELT is that you can store data in its original format, be it JSON, XML, PDF, or images. In ELT, you define the data's structure during the transformation phase, so you can use the source data in multiple downstream systems.

In an ELT process, data is extracted and loaded in its native format. This change reduces the time required to load the data into a destination system. The change also limits resource contention on the data sources.

The steps for the ELT process are the same as for the ETL process. They just follow a different order.

Another process like ELT is called extract, load, transform, and load (ELTL). The difference with ELTL is that it has a final load into a destination system.

There are two common types of data integration patterns that can be supported by Azure Data Factory.

Modern Data Warehouse workloads:

A Modern Data Warehouse is a centralized data store that provides descriptive analytics and decision support services across the whole enterprise using structured, unstructured, or streaming data sources. Data flows into the warehouse from multiple transactional systems, relational databases, and other data sources on a periodic basis. The stored data is used for historical and trend analysis reporting. The data warehouse acts as a central repository for many subject areas and contains the “single source of truth.”

Azure Data factory is typically used to automate the process of extracting, transforming, and loading the data through a batch process against structured and unstructured data sources.

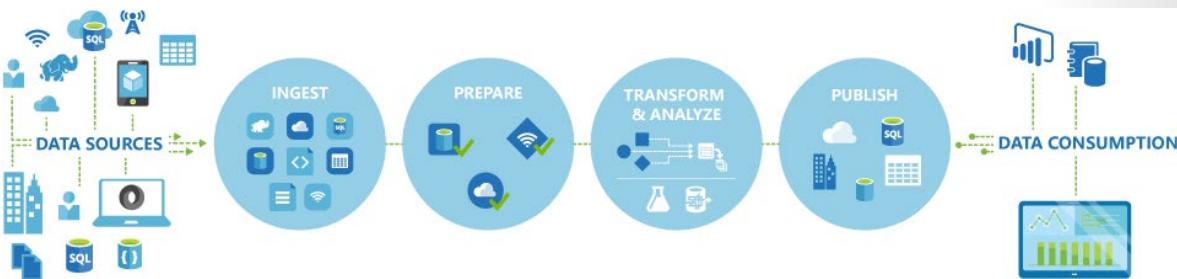
Advanced Analytical Workloads

You can perform advanced analytics in the form of predictive or preemptive analytics using a range of Azure data platform services. Azure Data Factory provides the integration from source systems into a Data Lake store, and can initiate compute resources such as Azure Databricks, or HDInsight to use the data to perform the advanced analytical work

Explain the data factory process

Data-driven workflows

The pipelines (data-driven workflows) in Azure Data Factory typically perform the following four steps:



Connect and collect

The first step in building an orchestration system is to define and connect all the required sources of data together, such as databases, file shares, and FTP web services. The next step is to ingest the data as needed to a centralized location for subsequent processing.

Transform and enrich

Compute services such as Databricks and Machine Learning can be used to prepare or produce transformed data on a maintainable and controlled schedule to feed production environments with cleansed and transformed data. In some instances, you may even augment the source data with additional data to aid analysis, or consolidate it through a normalization process to be used in a Machine Learning experiment as an example.

Publish

After the raw data has been refined into a business-ready consumable form from the transform and enrich phase, you can load the data into Azure Data Warehouse, Azure SQL Database, Azure Cosmos DB, or whichever analytics engine your business users can point to from their business intelligence tools

Monitor

Azure Data Factory has built-in support for pipeline monitoring via Azure Monitor, API, PowerShell, Azure Monitor logs, and health panels on the Azure portal, to monitor the scheduled activities and pipelines for success and failure rates.

Understand Azure Data Factory components

An Azure subscription might have one or more Azure Data Factory instances. Azure Data Factory is composed of four core components. These components work together to provide the platform on which you can compose data-driven workflows with steps to move and transform data.

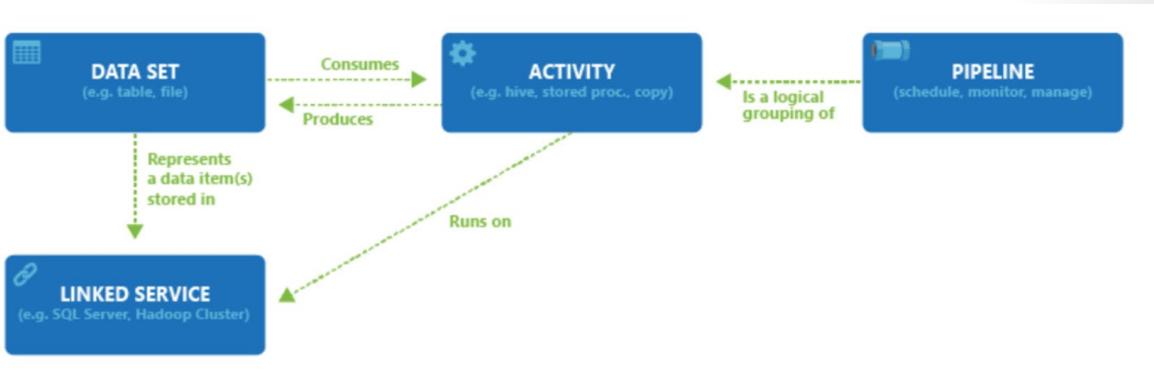


<https://www.microsoft.com/videoplayer/embed/RE4Mc3u>

Data Factory supports a wide variety of data sources that you can connect to through the creation of an object known as a **Linked Service**, which enables you to ingest the data from a data source in readiness to prepare the data for transformation and/or analysis. In addition, Linked Services can fire up compute services on demand. For example, you may have a requirement to start an on-demand HDInsight cluster for the purpose of just processing data through a Hive query. So Linked Services enables you to define data sources, or compute resource that is required to ingest and prepare data.

With the linked service defined, Azure Data Factory is made aware of the datasets that it should use through the creation of a **Datasets** object. Datasets represent data structures within the data store that is being referenced by the Linked Service object. Datasets can also be used by an ADF object known as an Activity.

Activities typically contain the transformation logic or the analysis commands of the Azure Data Factory's work. Activities includes the Copy Activity that can be used to ingest data from a variety of data sources. It can also include the Mapping Data Flow to perform code-free data transformations. It can also include the execution of a stored procedure, Hive Query, or Pig script to transform the data. You can push data into a Machine Learning model to perform analysis. It is not uncommon for multiple activities to take place that may include transforming data using a SQL stored procedure and then perform analytics with Databricks. In this case, multiple activities can be logically grouped together with an object referred to as a **Pipeline**, and these can be *scheduled* to execute, or a *trigger* can be defined that determines when a pipeline execution needs to be kicked off. There are different types of triggers for different types of events.



Control flow is an orchestration of pipeline activities that includes chaining activities in a sequence, branching, defining parameters at the pipeline level, and passing arguments while invoking the pipeline on-demand or from a trigger. It also includes custom-state passing and looping containers, and For-each iterators.

Parameters are key-value pairs of read-only configuration. Parameters are defined in the pipeline. The arguments for the defined parameters are passed during execution from the run context that was created by a trigger or a pipeline that was executed manually. Activities within the pipeline consume the parameter values.

Azure Data Factory has an *integration runtime* that enables it to bridge between the activity and linked Services objects. It is referenced by the linked service, and provides the compute environment where the activity either runs on or gets dispatched from. This way, the activity can be performed in the region closest possible. There are three types of Integration Runtime, including Azure, Self-hosted, and Azure-SSIS.

Once all the work is complete, you can then use Data Factory to publish the final dataset to another linked service that can then be consumed by technologies such as Power BI or Machine Learning.

Azure Data Factory security

To create Data Factory instances, the user account that you use to sign in to Azure must be a member of the *contributor* or *owner* role, or an *administrator* of the Azure subscription.

To create and manage Data Factory objects including datasets, linked services, pipelines, triggers, and integration runtimes, the following requirements must be met:

- To create and manage child resources in the Azure portal, you must belong to the *Data Factory Contributor* role at the resource group level or above.
- To create and manage resources with PowerShell or the SDK, the *contributor* role at the resource level or above is sufficient.

Data Factory Contributor role

When you are added as a member of this role, you have the following permissions:

- Create, edit, and delete data factories and child resources including datasets, linked services, pipelines, triggers, and integration runtimes.
- Deploy Resource Manager templates. Resource Manager deployment is the deployment method used by Data Factory in the Azure portal.

- Manage App Insights alerts for a data factory.
- At the resource group level or above, lets users deploy Resource Manager template.
- Create support tickets.

If the Data Factory Contributor role does not meet your requirement, you can create your own **custom role**¹.

Set-up Azure Data Factory

It is easy to set up Azure Data Factory from within the Azure portal, you only require the following information:

- **Name:** The name of the Azure Data Factory instance
- **Subscription:** The subscription in which the ADF instance is created
- **Resource group:** The resource group where the ADF instance will reside
- **Version:** select V2 for the latest features
- **Location:** The datacenter location in which the instance is stored

Enable Git provides the capability to integrate the code that you create with a Git repository enabling you to source control the code that you would create. Define the GIT url, repository name, branch name, and the root folder.

¹ <https://docs.microsoft.com/azure/role-based-access-control/custom-roles>

Home > New > Data Factory > New data factory

New data factory

Name *

Version ⓘ

V2

Subscription *

Resource Group *

Select existing...

Create new

Location * ⓘ

South Central US

Enable GIT ⓘ

GIT URL * ⓘ

Repo name * ⓘ

Branch Name * ⓘ

Root folder * ⓘ

Create

Alternatively, there are a number of different ways that you can provision the service programmatically. In this example you can see PowerShell at work to set up the environment.

```
#####
##          PART I: Creating an Azure Data Factory      ##
#####

# Sign in to Azure and set the WINDOWS AZURE subscription to work with
$SubscriptionId = "add your subscription in the quotes"

Add-AzureRmAccount
Set-AzureRmContext -SubscriptionId $SubscriptionId

# register the Microsoft Azure Data Factory resource provider
Register-AzureRmResourceProvider -ProviderNamespace Microsoft.DataFactory
```

```
# DEFINE RESOURCE GROUP NAME AND LOCATION PARAMETERS
$resourceGroupName = "cto_ignite"
$rglocation = "West US 2"

# CREATE AZURE DATA FACTORY
New-AzureRmDataFactoryV2 -ResourceGroupName $resourceGroupName -Name
"ctoigniteADF" -Location $rglocation
```

Create linked services

Before you create a dataset, you must create a **linked service** to link your data store to the data factory. Linked services are much like connection strings, which define the connection information needed for Data Factory to connect to external resources. There are over 100 connectors that can be used to define a linked service.

A linked service in Data Factory can be defined using the Copy Data Activity in the ADF designer, or you can create them independently to point to a data store or a compute resources. The Copy Activity copies data between the source and destination, and when you run this activity you are asked to define a linked service as part of the copy activity definition

Alternatively you can programmatically define a linked service in the JSON format to be used via REST APIs or the SDK, using the following notation:

```
{
    "name": "<Name of the linked service>",
    "properties": {
        "type": "<Type of the linked service>",
        "typeProperties": {
            "<data store or compute-specific type properties>"
        },
        "connectVia": {
            "referenceName": "<name of Integration Runtime>",
            "type": "IntegrationRuntimeReference"
        }
    }
}
```

The following table describes properties in the above JSON:

Property	Description	Required
name	Name of the linked service.	Yes
type	Type of the linked service. For example: AzureStorage (data store) or AzureBatch (compute). See the description for typeProperties.	Yes

Property	Description	Required
typeProperties	The type properties are different for each data store or compute. For the supported data store types and their type properties, see the dataset type table (https://docs.microsoft.com/azure/data-factory/concepts-datasets-linked-services#data-set-type). Navigate to the data store connector article to learn about type properties specific to a data store.	Yes
connectVia	The Integration Runtime (https://docs.microsoft.com/azure/data-factory/concepts-integration-runtime) to be used to connect to the data store. You can use Azure Integration Runtime or Self-hosted Integration Runtime (if your data store is located in a private network). If not specified, it uses the default Azure Integration Runtime.	No

Example of a Linked Service

Azure SQL Database

The following example creates a linked service named "AzureSqlLinkedService" that connects to an Azure SQL Database named "ctosqldb" with the userid of "ctest-a-oneill" and the password of "P@ssw0rd".

```
{
  "name": "AzureSqlLinkedService",
  "properties": {
    "type": "AzureSqlDatabase",
    "typeProperties": {
      "connectionString": "Server=tcp:<server-name>.database.windows.net,1433;Database=ctosqldb;User ID=ctest-a-oneill;Password=P@ssw0rd;Trusted_Connection=False;Encrypt=True;Connection Timeout=30"
    }
  }
}
```

Azure Blob Storage

The following example creates a linked service named "StorageLinkedService" that connects to an Azure Blob Store named "costorageaccount" with the storage account key used to connect to the data store

```
{  
    "name": "StorageLinkedService",  
    "properties": {  
        "type": "AzureStorage",  
        "typeProperties": {  
            "connectionString": "DefaultEndpointsProtocol=https;AccountName=ctos-  
storageaccount;AccountKey=<account-key>"  
        }  
    }  
}
```

Create datasets

A dataset is a named view of data that simply points or references the data you want to use in your activities as inputs and outputs. Datasets identify data within different data stores, such as tables, files, folders, and documents. For example, an Azure Blob dataset specifies the blob container and folder in Blob storage from which the activity should read the data.

A dataset in Data Factory can be defined as an object within the Copy Data Activity, as a separate object, or in a JSON format for programmatic creation as follows:

```
{  
    "name": "<name of dataset>",  
    "properties": {  
        "type": "<type of dataset: AzureBlob, AzureSql etc...>",  
        "linkedServiceName": {  
            "referenceName": "<name of linked service>",  
            "type": "LinkedServiceReference",  
        },  
        "schema": [  
            {  
                "name": "<Name of the column>",  
                "type": "<Name of the type>"  
            }  
        ],  
        "typeProperties": {  
            "<type specific property>": "<value>",  
            "<type specific property 2>": "<value 2>",  
        }  
    }  
}
```

The following table describes properties in the above JSON:

Property	Description	Required
name	Name of the dataset.	Yes
type	Type of the dataset. Specify one of the types supported by Data Factory (for example: AzureBlob, AzureSqlTable).	Yes

Property	Description	Required
structure	Schema of the dataset.	No
typeProperties	The type properties are different for each type (for example: Azure Blob, Azure SQL table).	Yes

Example of a dataset

Azure Blob

In this procedure, you create two datasets: InputDataset and OutputDataset. These datasets are of type Binary. They refer to the Azure Storage linked service named AzureStorageLinkedService. The input dataset represents the source data in the input folder. In the input dataset definition, you specify the blob container (adftutorial), the folder (input), and the file (emp.txt) that contain the source data. The output dataset represents the data that's copied to the destination. In the output dataset definition, you specify the blob container (adftutorial), the folder (output), and the file to which the data is copied.

1. Create a JSON file named InputDataset.json in the C:\ADFv2QuickStartPSH folder, with the following content:

```
{
  "name": "InputDataset",
  "properties": {
    "linkedServiceName": {
      "referenceName": "AzureStorageLinkedService",
      "type": "LinkedServiceReference"
    },
    "annotations": [],
    "type": "Binary",
    "typeProperties": {
      "location": {
        "type": "AzureBlobStorageLocation",
        "fileName": "emp.txt",
        "folderPath": "input",
        "container": "adftutorial"
      }
    }
  }
}
```

2. To create the dataset: InputDataset, run the Set-AzDataFactoryV2Dataset cmdlet.

```
Set-AzDataFactoryV2Dataset -DataFactoryName $DataFactory.DataFactoryName ` 
-ResourceGroupName $ResGrp.ResourceGroupName -Name "InputDataset" ` 
-DefinitionFile ".\InputDataset.json"
```

Here is the sample output:

```
DatasetName      : InputDataset
ResourceGroupName : <resourceGroupname>
```

```
DataFactoryName : <dataFactoryName>
Structure      :
Properties     : Microsoft.Azure.Management.DataFactory.Models.BinaryDataset
```

3. Repeat the steps to create the output dataset. Create a JSON file named OutputDataset.json in the C:\ADFv2QuickStartPSH folder, with the following content:

```
{
  "name": "OutputDataset",
  "properties": {
    "linkedServiceName": {
      "referenceName": "AzureStorageLinkedService",
      "type": "LinkedServiceReference"
    },
    "annotations": [],
    "type": "Binary",
    "typeProperties": {
      "location": {
        "type": "AzureBlobStorageLocation",
        "folderPath": "output",
        "container": "adftutorial"
      }
    }
  }
}
```

4. Run the Set-AzDataFactoryV2Dataset cmdlet to create the OutDataset.

```
Set-AzDataFactoryV2Dataset -DataFactoryName $DataFactory.DataFactoryName ` 
  -ResourceGroupName $ResGrp.ResourceGroupName -Name "OutputDataset" ` 
  -DefinitionFile ".\OutputDataset.json"
```

Here is the sample output:

```
DatasetName   : OutputDataset
ResourceGroupName : <resourceGroupName>
DataFactoryName : <dataFactoryName>
Structure      :
Properties     : Microsoft.Azure.Management.DataFactory.Models.BinaryDataset
```

Create data factory activities and pipelines

Activities within Azure Data Factory define the actions that will be performed on the data and there are three categories including:

- Data movement activities
- Data transformation activities
- Control activities

Data movement activities

Data movement activities simply move data from one data store to another. You can use the Copy Activity to perform data movement activities, or by using JSON. There are a wide range of data stores that are supported as a source and as a sink. This list is ever increasing, and you can find the **latest information here²**.

Data transformation activities

Data transformation activities can be performed natively within the authoring tool of Azure Data Factory using the Mapping Data Flow. Alternatively, you can call a compute resource to change or enhance data through transformation, or perform analysis of the data. These include compute technologies such as Azure Databricks, Azure Batch, SQL Database and Azure Synapse Analytics, Machine Learning Services, Azure Virtual machines and HDInsight. You can make use of any existing SQL Server Integration Services (SSIS) Packages stored in a Catalog to execute in Azure

As this list is always evolving, you can get the **latest information here³**.

Control activities

When graphically authoring ADF solutions, you can use the control flow within the designed to orchestrate pipeline activities that include chaining activities in a sequence, branching, defining parameters at the pipeline level, and passing arguments while invoking the pipeline on-demand or from a trigger. The current capabilities include:

Control Activity	Description
Execute Pipeline Activity	Execute Pipeline activity allows a Data Factory pipeline to invoke another pipeline.
ForEachActivity	ForEach Activity defines a repeating control flow in your pipeline. This activity is used to iterate over a collection and executes specified activities in a loop. The loop implementation of this activity is similar to Foreach looping structure in programming languages.
WebActivity	Web Activity can be used to call a custom REST endpoint from a Data Factory pipeline. You can pass datasets and linked services to be consumed and accessed by the activity.
Lookup Activity	Lookup Activity can be used to read or look up a record/ table name/ value from any external source. This output can further be referenced by succeeding activities.
Get Metadata Activity	GetMetadata activity can be used to retrieve metadata of any data in Azure Data Factory.

² <https://docs.microsoft.com/azure/data-factory/concepts-pipelines-activities#data-movement-activities>

³ <https://docs.microsoft.com/azure/data-factory/concepts-pipelines-activities#data-transformation-activities>

Control Activity	Description
Until Activity	Implements Do-Until loop that is similar to Do-Until looping structure in programming languages. It executes a set of activities in a loop until the condition associated with the activity evaluates to true. You can specify a timeout value for the until activity in Data Factory.
If Condition Activity	The If Condition can be used to branch based on condition that evaluates to true or false. The If Condition activity provides the same functionality that an if statement provides in programming languages. It evaluates a set of activities when the condition evaluates to true and another set of activities when the condition evaluates to false.
Wait Activity	When you use a Wait activity in a pipeline, the pipeline waits for the specified period of time before continuing with execution of subsequent activities.

You can get the [latest information here⁴](#).

Activities and pipelines

Defining activities

When using JSON notation, the activities section can have one or more activities defined within it. There are two main types of activities: Execution and Control Activities. Execution (also known as Compute) activities include data movement and data transformation activities. They have the following top-level structure:

```
{  
    "name": "Execution Activity Name",  
    "description": "description",  
    "type": "<ActivityType>",  
    "typeProperties":  
    {  
    },  
    "linkedServiceName": "MyLinkedService",  
    "policy":  
    {  
    },  
    "dependsOn":  
    {  
    }  
}
```

The following table describes properties in the above JSON:

⁴ <https://docs.microsoft.com/azure/data-factory/concepts-pipelines-activities#control-activities>

Property	Description	Required
name	Name of the activity.	Yes
description	Text describing what the activity or is used for.	Yes
type	Defines the type of the activity.	Yes
linkedServiceName	Name of the linked service used by the activity.	Yes for HDInsight, Machine Learning Batch Scoring Activity and Stored Procedure Activity
typeProperties	Properties in the typeProperties section depend on each type of activity.	No
policy	Policies that affect the run-time behavior of the activity. This property includes timeout and retry behavior.	No
dependsOn	This property is used to define activity dependencies, and how subsequent activities depend on previous activities.	No

Defining control activities

A Control Activity in Data Factory is defined in JSON format as follows:

```
{
    "name": "Control Activity Name",
    "description": "description",
    "type": "<ActivityType>",
    "typeProperties":
    {
    },
    "dependsOn":
    {
    }
}
```

The following table describes properties in the above JSON:

Property	Description	Required
name	Name of the activity.	Yes
description	Text describing what the activity or is used for.	Yes
type	Defines the type of the activity.	Yes
typeProperties	Properties in the typeProperties section depend on each type of activity.	No

Property	Description	Required
dependsOn	This property is used to define activity dependencies, and how subsequent activities depend on previous activities.	No

Defining pipelines

Here is how a pipeline is defined in JSON format:

```
{
  "name": "PipelineName",
  "properties":
  {
    "description": "pipeline description",
    "activities":
    [
    ],
    "parameters": {
    }
  }
}
```

The following table describes properties in the above JSON:

Property	Description	Required
name	Name of the pipeline.	Yes
description	Text describing what the pipeline is used for.	No
activities	The activities section can have one or more activities defined within it.	Yes
parameters	The parameters section can have one or more parameters defined within the pipeline, making your pipeline flexible for reuse.	No

Example

The following JSON defines pipeline named "MyFirstPipeline" that contains one activity type of HDInsightHive that will call a query from a script name "partitionweblogs.hql" that is stored in the linked service named "StorageLinkedService", with an input named "AzureBlobInput" and an output named "AzureBlobOutput". It executes this against the compute resource defined in the linked service named "HDInsightOnDemandLinkedService"

The pipeline is scheduled to execute on a monthly basis, and will attempt to execute 3 times should it fail.

```
{
  "name": "MyFirstPipeline",
  "properties": {
    "description": "My first Azure Data Factory pipeline",
```

```
    "activities": [
        {
            "type": "HDInsightHive",
            "typeProperties": {
                "scriptPath": "adfgetstarted/script/partitionweblogs.hql",
                "scriptLinkedService": "StorageLinkedService",
                "defines": {
                    "inputtable": "wasb://adfgetstarted@ctostorageaccount.blob.core.windows.net/inputdata",
                    "partitionedtable": "wasb://adfgetstarted@ctostorageaccount.blob.core.windows.net/partitioneddata"
                }
            },
            "inputs": [
                {
                    "name": "AzureBlobInput"
                }
            ],
            "outputs": [
                {
                    "name": "AzureBlobOutput"
                }
            ],
            "policy": {
                "concurrency": 1,
                "retry": 3
            },
            "scheduler": {
                "frequency": "Month",
                "interval": 1
            },
            "name": "RunSampleHiveActivity",
            "linkedServiceName": "HDInsightOnDemandLinkedService"
        }
    ],
    "start": "2017-04-01T00:00:00Z",
    "end": "2017-04-02T00:00:00Z",
    "isPaused": false,
    "hubName": "ctogetstartedddf_hub",
    "pipelineMode": "Scheduled"
}
```

Manage integration runtimes

In Data Factory, an activity defines the action to be performed. A linked service defines a target data store or a compute service. An integration runtime provides the infrastructure for the activity and linked services.

Integration Runtime is referenced by the linked service or activity, and provides the compute environment where the activity either runs on or gets dispatched from. This way, the activity can be performed in the region closest possible to the target data store or compute service in the most performant way while meeting security and compliance needs.

In short, the Integration Runtime (IR) is the compute infrastructure used by Azure Data Factory. It provides the following data integration capabilities across different network environments, including:

- **Data Flow:** Execute a Data Flow in managed Azure compute environment.
- **Data movement:** Copy data across data stores in public network and data stores in private network (on-premises or virtual private network). It provides support for built-in connectors, format conversion, column mapping, and performant and scalable data transfer.
- **Activity dispatch:** Dispatch and monitor transformation activities running on a variety of compute services such as Azure Databricks, Azure HDInsight, Azure Machine Learning, Azure SQL Database, SQL Server, and more.
- **SSIS package execution:** Natively execute SQL Server Integration Services (SSIS) packages in a managed Azure compute environment.

Whenever an Azure Data Factory instance is created, a default Integration Runtime environment is created that supports operations on cloud data stores and compute services in public network. This can be viewed when the integration runtime is set to Auto-Resolve

Integration runtime types

Data Factory offers three types of Integration Runtime, and you should choose the type that best serve the data integration capabilities and network environment needs you are looking for. These three types are:

- Azure
- Self-hosted
- Azure-SSIS

You can explicitly define the Integration Runtime setting in the **connectVia** property, if this is not defined, then the default Integration Runtime is used with the property set to Auto-Resolve.

The following table describes the capabilities and network support for each of the integration runtime types:

IR type	Public network	Private network
Azure	Data Flow Data movement Activity dispatch	
Self-hosted	Data movement Activity dispatch	Data movement Activity dispatch
Azure-SSIS	SSIS package execution	SSIS package execution

Determining which integration runtime to use

There are a range of factors that affect the Integration Runtime that you will use. The following is a guide that will help you select the right IR

Copy activity

For the Copy activity, it requires source and sink linked services to define the direction of data flow. The following logic is used to determine which integration runtime instance is used to perform the copy:

- Copying between two cloud data sources: when both source and sink linked services are using Azure IR, ADF will use the regional Azure IR if you specified, or auto determine a location of Azure IR if you choose the auto-resolve IR (default) as described in Integration runtime location section.
- Copying between a cloud data source and a data source in private network: if either source or sink linked service points to a self-hosted IR, the copy activity is executed on that self-hosted Integration Runtime.
- Copying between two data sources in private network: both the source and sink Linked Service must point to the same instance of integration runtime, and that integration runtime is used to execute the copy Activity.

Lookup and GetMetadata activity

The Lookup and GetMetadata activity is executed on the integration runtime associated to the data store linked service.

Transformation activity

Each transformation activity has a target compute Linked Service, which points to an integration runtime. This integration runtime instance is where the transformation activity is dispatched from.

Data Flow activity

Data Flow activity is executed on the integration runtime associated to it.

Knowledge check

Question 1

Which Azure Data Factory component orchestrates a transformation job or runs a data movement command?

- Linked Services
- Datasets
- Activities

Question 2

You are moving data from an Azure Data Lake Gen2 store to Azure Synapse Analytics. Which Azure Data Factory integration runtime would be used in a data copy activity?

- Azure-SSIS
- Azure
- Self-hosted

Summary

In this section, you have learned how Azure Data Factory can be used to manage a regular process of data movement as part of a wider enterprise analytical solution. In this introductory section you have learned how Azure Data Factory meets that need through its component parts and the Azure Data Factory process. You also now understand the security groups required to provision and manage the service.

Important: Should you be working in your own subscription while running through this content, it is best practice at the end of a project to identify whether or not you still need the resources you created. Resources left running can cost you money. You can delete resources one by one, or just delete the resource group to get rid of the entire set.

Perform code-free transformation at scale with Azure Data Factory

Introduction

Regardless of the skillset that you possess to manipulate data, Azure Data Factory provides an array of tools that enables you to transform and cleanse data to your business requirements.

Many common transformation tasks can be performed code free, using the Mapping Data Flow task. You can also make use of other Azure Data Services to perform the work, and even make use of existing SQL Server Integration Packages that you have.

Azure Data Factory also enables you to do data exploration using Wrangling Data Flows.

Learning objectives

In this module, you will:

- Explain transformation methods
- Describe transformation types
- Use Mapping Data Flow
- Debug Mapping Data Flow
- Use Wrangling Data
- Use compute transformations with Azure Data Factory
- Integrate SSI packages with Azure Data Factory

Explain Azure Data Factory transformation methods

Just as Azure Data Factory provides a variety of methods for ingesting data, it also provides a range of methods to perform transformations. You can pick a method that matches the skillsets of your team or takes advantage of existing technologies that you already have in your data estate. There is also the opportunity to perform transformations without writing code at all using the Mapping Data Flow.



<https://www.microsoft.com/videoplayer/embed/RE4MeGO>

Transforming data using Mapping Data Flow

Mapping Data Flows provide an environment for building a wide range of data transformations visually without the need to use code. The resulting data flows that are created are subsequently executed on scaled-out Apache Spark clusters that are automatically provisioned when you execute the Mapping Data Flow. Mapping Data Flows also provides the capability to monitor the execution of the transformations so that you can view how the transformations are progressing, or to understand any errors that may occur.

Transforming data using compute resources

Azure Data Factory can also call on compute resources to transform data by a data platform service that may be better suited to the job. A great example of this is that Azure Data Factory can create a pipeline to an analytical data platform such as Spark pools in an Azure Synapse Analytics instance to perform a complex calculation using python. Another example could be to send data to an Azure SQL Database instance to execute a stored procedure using Transact-SQL. There is a wide range of compute resource, and the associated activities that they can perform as shown in the following table:

Compute environment	activities
On-demand HDInsight cluster or your own HDInsight cluster	Hive, Pig, Spark, MapReduce, Hadoop Streaming
Azure Batch	Custom activities
Azure Machine Learning Studio Machine	Learning activities: Batch Execution and Update Resource
Azure Machine Learning	Azure Machine Learning Execute Pipeline
Azure Data Lake Analytics	Data Lake Analytics U-SQL
Azure SQL, Azure SQL Data Warehouse, SQL Server	Stored Procedure
Azure Databricks	Notebook, Jar, Python
Azure Function	Azure Function activity

Transforming data using SQL Server Integration Services (SSIS) packages

Many organizations have decades of development investment in SSIS packages that contain both ingestion and transformation logic from on-premises and cloud data stores. Azure Data Factory provides the ability to lift and shift existing SSIS workload, by creating an Azure-SSIS Integration Runtime to natively execute SSIS packages. Using Azure-SSIS Integration Runtime will enable you to deploy and manage your existing SSIS packages with little to no change using familiar tools such as SQL Server Data Tools (SSDT) and SQL Server Management Studio (SSMS), just like using SSIS on premises.

Describe Azure Data Factory transformation types

Mapping Data Flows provides a number of different transformations types that enable you to modify data. They are broken down into the following categories:

Category Name	Description
Schema modifier transformations	These types of transformations will make a modification to a sink destination by creating new columns based on the action of the transformation. An example of this is the Derived Column transformation that will create a new column based on the operations performed on existing column.
Row modifier transformations	These types of transformations impact how the rows are presented in the destination. An example of this is a Sort transformation that orders the data.

Category Name	Description
Multiple inputs/outputs transformations	These types of transformations will generate new data pipelines or merge pipelines into one. An example of this is the Union transformation that combines multiple data streams.

Below is a list of transformations that is available in the Mapping Data Flows

Name	Category	Description
Aggregate	Schema modifier	Define different types of aggregations such as SUM, MIN, MAX, and COUNT grouped by existing or computed columns.
Alter row	Row modifier	Set insert, delete, update, and upsert policies on rows. You can add one-to-many conditions as expressions. These conditions should be specified in order of priority, as each row will be marked with the policy corresponding to the first-matching expression. Each of those conditions can result in a row (or rows) being inserted, updated, deleted, or upserted. Alter Row can produce both DDL & DML actions against your database.
Conditional split	Multiple inputs/outputs	Route rows of data to different streams based on matching conditions.
Derived column	Schema modifier	generate new columns or modify existing fields using the data flow expression language.
Exists	Multiple inputs/outputs	Check whether your data exists in another source or stream.
Filter	Row modifier	Filter a row based upon a condition.
Flatten	Schema modifier	Take array values inside hierarchical structures such as JSON and unroll them into individual rows.
Join	Multiple inputs/outputs	Combine data from two sources or streams.
Lookup	Multiple inputs/outputs	Enables you to reference data from another source.
New branch	Multiple inputs/outputs	Apply multiple sets of operations and transformations against the same data stream.

Name	Category	Description
Pivot	Schema modifier	An aggregation where one or more grouping columns has distinct row values transformed into individual columns.
Select	Schema modifier	Alias columns and stream names, and drop or reorder columns.
Sink	-	A final destination for your data.
Sort	Row modifier	Sort incoming rows on the current data stream.
Source	-	A data source for the data flow.
Surrogate key	Schema modifier	Add an incrementing non-business arbitrary key value.
Union	Multiple inputs/outputs	Combine multiple data streams vertically.
Unpivot	Schema modifier	Pivot columns into row values.
Window	Schema modifier	Define window-based aggregations of columns in your data streams.

Data Flow Expression Builder

Some of the transformations that you can define have an **Data Flow Expression Builder** that will enable you to customize the functionality of a transformation using columns, fields, variables, parameters, functions from your data flow in these boxes.

To build the expression, use the Expression Builder, which is launched by clicking in the expression text box inside the transformation. You'll also sometimes see "Computed Column" options when selecting columns for transformation. When you click that, you'll also see the Expression Builder launched.

Visual expression builder

FUNCTIONS

All Functions Input schema Parameters

abc movie
abc title
abc genres

year <= 1920

+

Data preview

Output:	year
✓	1920
✗	1921
✓	1920
✗	1921

Save and finish Cancel Clear contents

The Expression Builder tool defaults to the text editor option. the auto-complete feature reads from the entire Azure Data Factory Data Flow object model with syntax checking and highlighting.

Author an Azure Data Factory mapping data flow

Transforming data with the Mapping Data Flow

You can natively perform data transformations with Azure Data Factory code free using the Mapping Data Flow task. Mapping Data Flows provide a fully visual experience with no coding required. Your data flows will run on your own execution cluster for scaled-out data processing. Data flow activities can be operationalized via existing Data Factory scheduling, control, flow, and monitoring capabilities.

When building data flows, you can enable debug mode, which turns on a small interactive Spark cluster. Turn on debug mode by toggling the slider at the top of the authoring module. Debug clusters take a few minutes to warm up, but can be used to interactively preview the output of your transformation logic.



With the Mapping Data Flow added, and the Spark cluster running, this will enable you to perform the transformation, and run and preview the data. No coding is required as Azure Data Factory handles all the code translation, path optimization, and execution of your data flow jobs.

Adding source data to the Mapping Data Flow

Open the Mapping Data Flow canvas. Click on the Add Source button in the Data Flow canvas. In the source dataset dropdown, select your data source, in this case the ADLS Gen2 dataset is used in this example

The screenshot shows the Azure Mapping Data Flow interface. At the top, there's a preview pane showing a single row from the 'MoviesADLS' dataset, which has 0 total columns. Below this is a dashed box labeled 'Add Source'. The main area contains several tabs: 'Source Settings' (which is selected), 'Source Options', 'Projection', 'Optimize', 'Inspect', and 'Data Preview'. Under 'Source Settings', there are fields for 'Output stream name' (set to 'MoviesADLS'), 'Source dataset' (set to 'DelimitedText1'), and various 'Options' like 'Allow schema drift' (checked) and 'Infer drifted column types'. There's also a 'Skip line count' field and a 'Sampling' section with a radio button set to 'Disable'.

There are a couple of points to note:

- If your dataset is pointing at a folder with other files and you only want to use one file, you may need to create another dataset or utilize parameterization to make sure only a specific file is read
- If you have not imported your schema in your ADLS, but have already ingested your data, go to the dataset's 'Schema' tab and click 'Import schema' so that your data flow knows the schema projection.

Mapping Data Flow follows an extract, load, transform (ELT) approach and works with staging datasets that are all in Azure. Currently the following datasets can be used in a source transformation:

- Azure Blob Storage (JSON, Avro, Text, Parquet)
- Azure Data Lake Storage Gen1 (JSON, Avro, Text, Parquet)
- Azure Data Lake Storage Gen2 (JSON, Avro, Text, Parquet)
- Azure Synapse Analytics
- Azure SQL Database
- Azure CosmosDB

Azure Data Factory has access to over 80 native connectors. To include data from those other sources in your data flow, use the Copy Activity to load that data into one of the supported staging areas.

Once your debug cluster is warmed up, verify your data is loaded correctly via the Data Preview tab. Once you click the refresh button, Mapping Data Flow will show a snapshot of what your data looks like when it is at each transformation.

The screenshot shows the Data Preview tab in Azure Data Studio. At the top, there's a summary for 'MoviesADLS' with '6 total' columns. Below that is an 'Add Column' button. On the right, there are buttons for adding (+), removing (-), and filtering (🔍). The main area displays a preview of movie data with the following columns and rows:

	movie	title	genres	year	Rating	Rotten Tomat
+	108583	Fawlty Towers (1975)	Comedy	-1980	1	54
+	32898	Trip to the Moon, A (Voyage ...	Action Adventure Fantasy Sci...	1902	7	80
+	7065	Birth of a Nation, The	Drama War	1915	6	92
+	7243	Intolerance: Love's Struggle ...	Drama	1915	4	82
+	62383	20,000 Leagues Under the Sea	Action Adventure Sci-Fi	1915	9	92
+	8511	Immigrant, The	Comedy	1917	4	59
+	3309	Dog's Life, A	Comedy	1917	3	83

Using transformations in the Mapping Data Flow

Now that you have moved the data into Azure Data Lake Store Gen2, you are ready to build a Mapping Data Flow that will transform your data at scale via a spark cluster and then load it into a Data Warehouse.

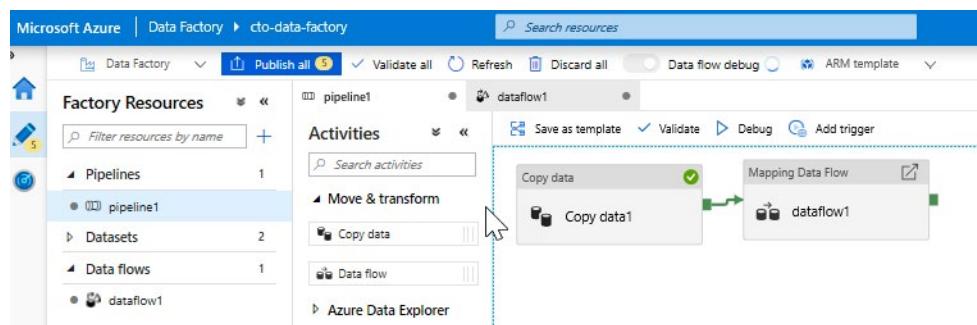
The main tasks for this are as follows:

1. Preparing the environment
2. Adding a Data Source
3. Using Mapping Data Flow transformation
4. Writing to a Data Sink

Task 1: Preparing the environment

1. **Turn on Data Flow Debug** Turn the **Data Flow Debug** slider located at the top of the authoring module on.

NOTE: Data Flow clusters take 5-7 minutes to warm up.
2. **Add a Data Flow activity.** In the Activities pane, open the Move and Transform accordion and drag the **Data Flow** activity onto the pipeline canvas. In the blade that pops up, click **Create new Data Flow** and select **Mapping Data Flow** and then click **OK**. Click on the **Pipeline1** tab and drag the green box from your Copy activity to the Data Flow Activity to create an on success condition. You will see the following in the canvass:



Task 2: Adding a Data Source

1. **Add an ADLS source.** Double-click on the Mapping Data Flow object in the canvas. Click on the Add Source button in the Data Flow canvas. In the **Source dataset** dropdown, select your **ADLSG2** dataset used in your Copy activity

The screenshot shows the 'Source Settings' tab of the Azure Data Flow configuration. The 'Source dataset' dropdown is set to 'DelimitedText1'. Other settings include 'Output stream name' (MoviesADLS), 'Allow schema drift' (checked), and 'Sampling' (Disable selected). The 'Source Options' tab is also visible.

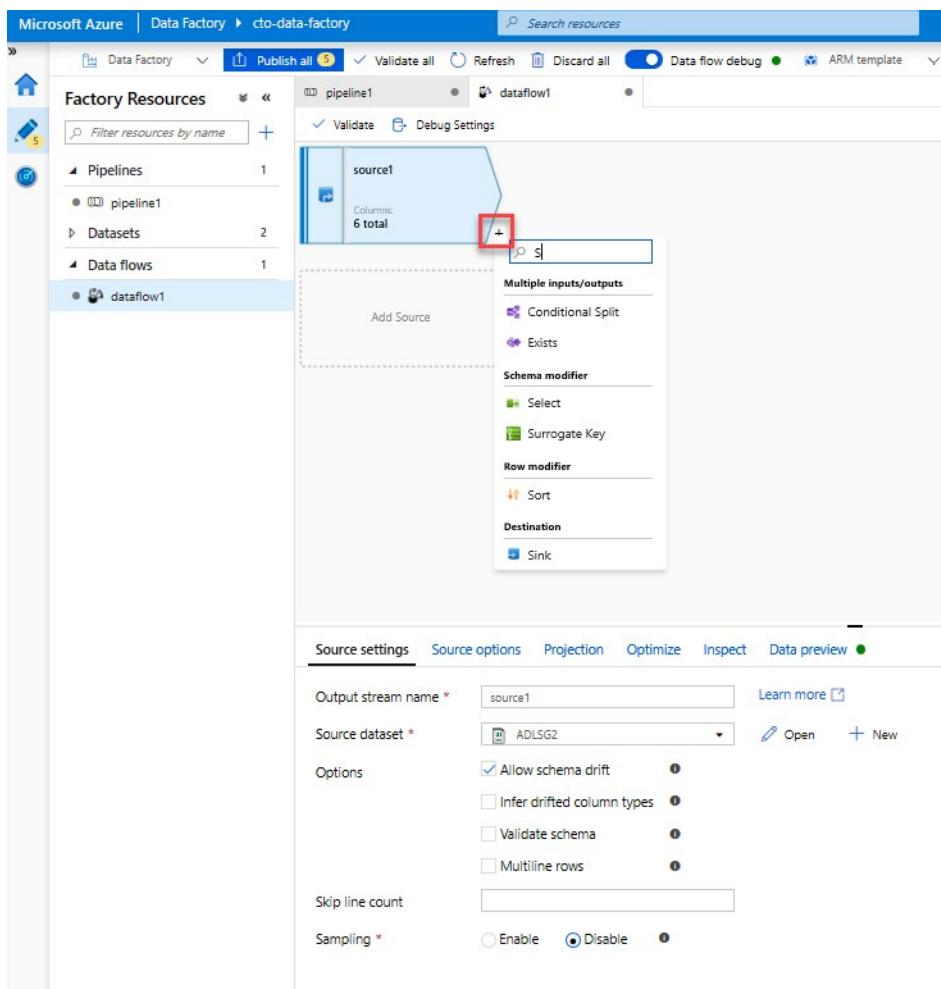
Setting	Value
Output stream name *	MoviesADLS
Source dataset *	DelimitedText1
Options	<input checked="" type="checkbox"/> Allow schema drift <input type="checkbox"/> Infer drifted column types <input type="checkbox"/> Validate schema
Skip line count	[Input field]
Sampling *	<input type="radio"/> Enable <input checked="" type="radio"/> Disable

- If your dataset is pointing at a folder with other files, you may need to create another dataset or utilize parameterization to make sure only the moviesDB.csv file is read
- If you have not imported your schema in your ADLS, but have already ingested your data, go to the dataset's 'Schema' tab and click 'Import schema' so that your data flow knows the schema projection.

Once your debug cluster is warmed up, verify your data is loaded correctly via the Data Preview tab. Once you click the refresh button, Mapping Data Flow will show a snapshot of what your data looks like when it is at each transformation.

Task 3: Using Mapping Data Flow transformation

1. **Add a Select transformation to rename and drop a column.** In the preview of the data, you may have noticed that the “Rotten Tomatoes” column is misspelled. To correctly name it and drop the unused Rating column, you can add a **Select transformation⁵** by clicking on the + icon next to your ADLS source node and choosing Select under Schema modifier.



In the **Name** as field, change 'Rotton' to 'Rotten'. To drop the Rating column, hover over it and click on the trash can icon.

⁵ <https://docs.microsoft.com/azure/data-factory/data-flow-select>

2. **Add a Filter Transformation to filter out unwanted years.** Say you are only interested in movies made after 1951. You can add a **Filter transformation**⁶ to specify a filter condition by clicking on the **+ icon** next to your Select transformation and choosing **Filter** under Row Modifier. Click on the **expression box** to open up the **Expression builder**⁷ and enter in your filter condition. Using the syntax of the **Mapping Data Flow expression language**⁸, `toInteger(year) > 1950` will convert the string year value to an integer and filter rows if that value is above 1950.

You can use the expression builder's embedded Data preview pane to verify your condition is working properly

3. **Add a Derive Transformation to calculate primary genre.** As you may have noticed, the genres column is a string delimited by a ' | ' character. If you only care about the *first* genre in each column,

⁶ <https://docs.microsoft.com/azure/data-factory/data-flow-filter>

⁷ <https://docs.microsoft.com/azure/data-factory/concepts-data-flow-expression-builder>

⁸ <https://docs.microsoft.com/azure/data-factory/data-flow-expression-functions>

you can derive a new column named **PrimaryGenre** via the **Derived Column⁹** transformation by clicking on the **+** icon next to your Filter transformation and choosing Derived under Schema Modifier. Similar to the filter transformation, the derived column uses the Mapping Data Flow expression builder to specify the values of the new column.

The screenshot shows the 'Derived column's settings' pane. The 'Output stream name' is set to 'DerivedPrimaryGenre'. The 'Incoming stream' is 'Filter1'. In the 'Columns' section, there is one column 'PrimaryGenre' with the expression 'iif(locate('|',genres) > 1, left(genres, locate('|',genres)-1), genres)'.

In this scenario, you are trying to extract the first genre from the genres column, which is formatted as 'genre1|genre2|...|genreN'. Use the **locate** function to get the first 1-based index of the '|' in the genres string. Using the **iif** function, if this index is greater than 1, the primary genre can be calculated via the **left** function, which returns all characters in a string to the left of an index. Otherwise, the PrimaryGenre value is equal to the genres field. You can verify the output via the expression builder's Data preview pane.

4. **Rank movies via a Window Transformation.** Say you are interested in how a movie ranks within its year for its specific genre. You can add a **Window transformation¹⁰** to define window-based aggregations by clicking on the **+** icon next to your Derived Column transformation and clicking Window under Schema modifier. To accomplish this, specify what you are windowing over, what you are sorting by, what the range is, and how to calculate your new window columns. In this example, we will window over PrimaryGenre and year with an unbounded range, sort by Rotten Tomato descending, and calculate a new column called RatingsRank that is equal to the rank each movie has within its specific genre-year.

The screenshot shows the 'Window Settings' pane. The 'Output stream name' is 'RankMoviesByRatings' and the 'Incoming stream' is 'DerivePrimaryGenre'. The steps are: 1. Over, 2. Sort, 3. Range by, 4. Window columns. Under 'Step 4', there are two columns: 'PrimaryGenre' and 'year'. The 'Name as' fields are also shown.

DerivePrimaryGenre's column	Name as
PrimaryGenre	PrimaryGenre
year	year

⁹ <https://docs.microsoft.com/azure/data-factory/data-flow-derived-column>

¹⁰ <https://docs.microsoft.com/azure/data-factory/data-flow-window>

The figure consists of three vertically stacked screenshots of the Azure Data Factory Data Flow 'Window Settings' interface. Each screenshot shows a top navigation bar with 'Window Settings', 'Optimize', 'Inspect', and 'Data Preview'. Below this, there are sections for 'Output stream name' (set to 'RankMoviesByRatings') and 'Incoming stream' (set to 'DerivePrimaryGenre'). A horizontal tab bar at the bottom allows switching between four steps: 1. Over, 2. Sort, 3. Range by, and 4. Window columns. In the first screenshot, '2. Sort' is selected, showing a dropdown menu for 'DerivePrimaryGenre's column' containing 'abc Rotten Tomato' with an arrow pointing down, and checkboxes for 'Order' (unchecked) and 'Nulls first' (checked). In the second screenshot, '3. Range by' is selected, showing options for 'Option' (radio buttons for 'Range by current row offset' and 'Range by column value', with the former selected), and a checkbox for 'Unbounded' which is checked. In the third screenshot, '4. Window columns' is selected, showing a dropdown menu for 'RatingsRank' with an arrow pointing down, and a text input field containing 'rank()' with a small '123' icon to its right.

5. **Aggregate ratings with an Aggregate Transformation.** Now that you have gathered and derived all your required data, we can add an **Aggregate transformation**¹¹ to calculate metrics based on a desired group by clicking on the **+** icon next to your Window transformation and clicking Aggregate under Schema modifier. As you did in the window transformation, lets group movies by PrimaryGenre and year

¹¹ <https://docs.microsoft.com/azure/data-factory/data-flow-aggregate>

The screenshot shows the 'Aggregate settings' interface with the 'Aggregates' tab selected. It includes fields for 'Output stream name' (AggregateRatings), 'Incoming stream' (RankMoviesByRatings), and 'Group by' (PrimaryGenre, year). The 'Name as' section maps these to new columns: PrimaryGenre and year.

In the Aggregates tab, you can aggregations calculated over the specified group by columns. For every genre and year, lets get the average Rotten Tomatoes rating, the highest and lowest rated movie (utilizing the windowing function) and the number of movies that are in each group. Aggregation significantly reduces the number of rows in your transformation stream and only propagates the group by and aggregate columns specified in the transformation.

The screenshot shows the 'Aggregate settings' interface with the 'Aggregates' tab selected. It includes fields for 'Output stream name' (AggregateRatings), 'Incoming stream' (RankMoviesByRatings), and 'Group by' (PrimaryGenre, year). The 'Aggregates' section contains four entries:

- AverageRating: avg(toInteger({Rotten Tomato}))
- HighestRated: first(title)
- LowestRated: last(title)
- NumberOfMovies: count()

- To see how the aggregate transformation changes your data, use the Data Preview tab
6. **Specify Upsert condition via an Alter Row Transformation.** If you are writing to a tabular sink, you can specify insert, delete, update and upsert policies on rows using the **Alter Row transformation**¹² by clicking on the + icon next to your Aggregate transformation and clicking Alter Row under Row modifier. Since you are always inserting and updating, you can specify that all rows will always be upserted.

The screenshot shows the 'Alter row settings' interface. It includes fields for 'Output stream name' (UpsertIfTrue), 'Incoming stream' (AggregateRatings), and 'Alter row conditions' (Upssert if true).

¹² <https://docs.microsoft.com/azure/data-factory/data-flow-alter-row>

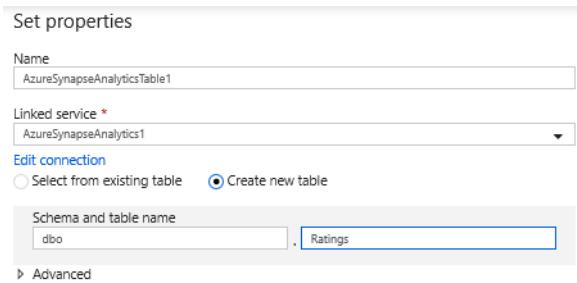
Task 4: Writing to a Data Sink

1. **Write to a Azure Synapse Analytics Sink.** Now that you have finished all your transformation logic, you are ready to write to a Sink.

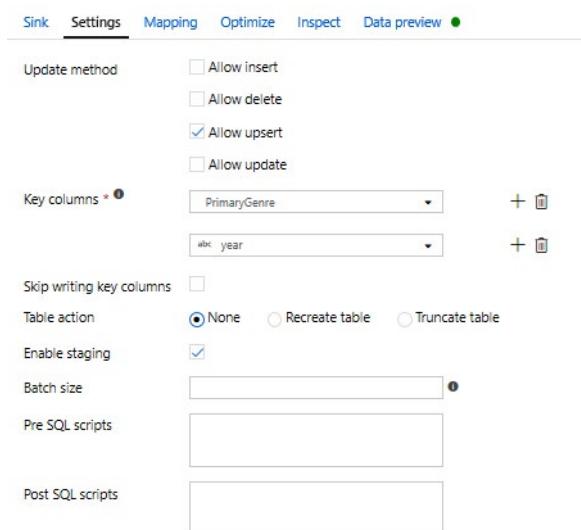
1. Add a **Sink** by clicking on the **+** icon next to your Upsert transformation and clicking Sink under Destination.
2. In the Sink tab, create a new data warehouse dataset via the **+ New button**.
3. Select **Azure Synapse Analytics** from the tile list.
4. Select a new linked service and configure your Azure Synapse Analytics connection to connect to the DWDB database created in Module 5. Click **Create** when finished.

The screenshot shows the 'New linked service' configuration dialog for Azure Synapse Analytics. The 'Name' field is set to 'AzureSynapseAnalytics1'. The 'Description' field is empty. Under 'Connect via integration runtime *', 'AutoResolveIntegrationRuntime' is selected. The 'Connection string' tab is active, showing 'Account selection method' set to 'From Azure subscription' (radio button selected). The 'Azure Key Vault' tab is also visible. Below this, 'Azure subscription' is set to 'dwhservicecto', 'Server name' is 'dwhservicecto', 'Database name' is 'DWDB', 'Authentication type' is 'SQL authentication', 'User name' is 'ctosqladmin', and 'Password' is masked. At the bottom, there are sections for 'Additional connection properties', 'Annotations', and 'Parameters'.

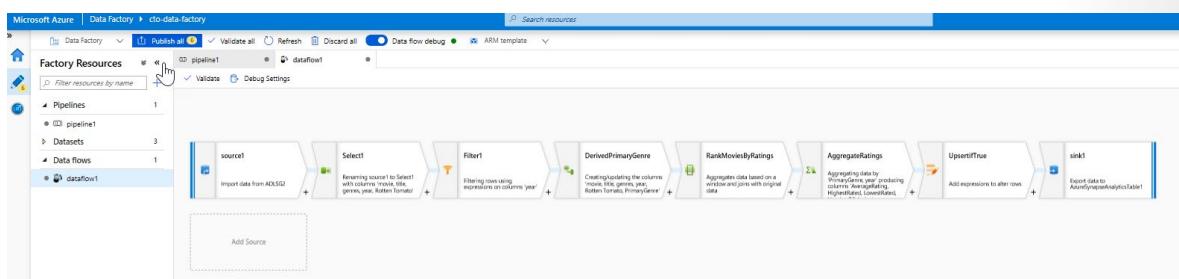
5. In the dataset configuration, select **Create new table** and enter in the schema of **Dbo** and the table name of **Ratings**. Click **OK** once completed.



- Since an upsert condition was specified, you need to go to the Settings tab and select 'Allow upsert' based on key columns PrimaryGenre and year.



At this point, You have finished building your 8 transformation Mapping Data Flow. It's time to run the pipeline and see the results!

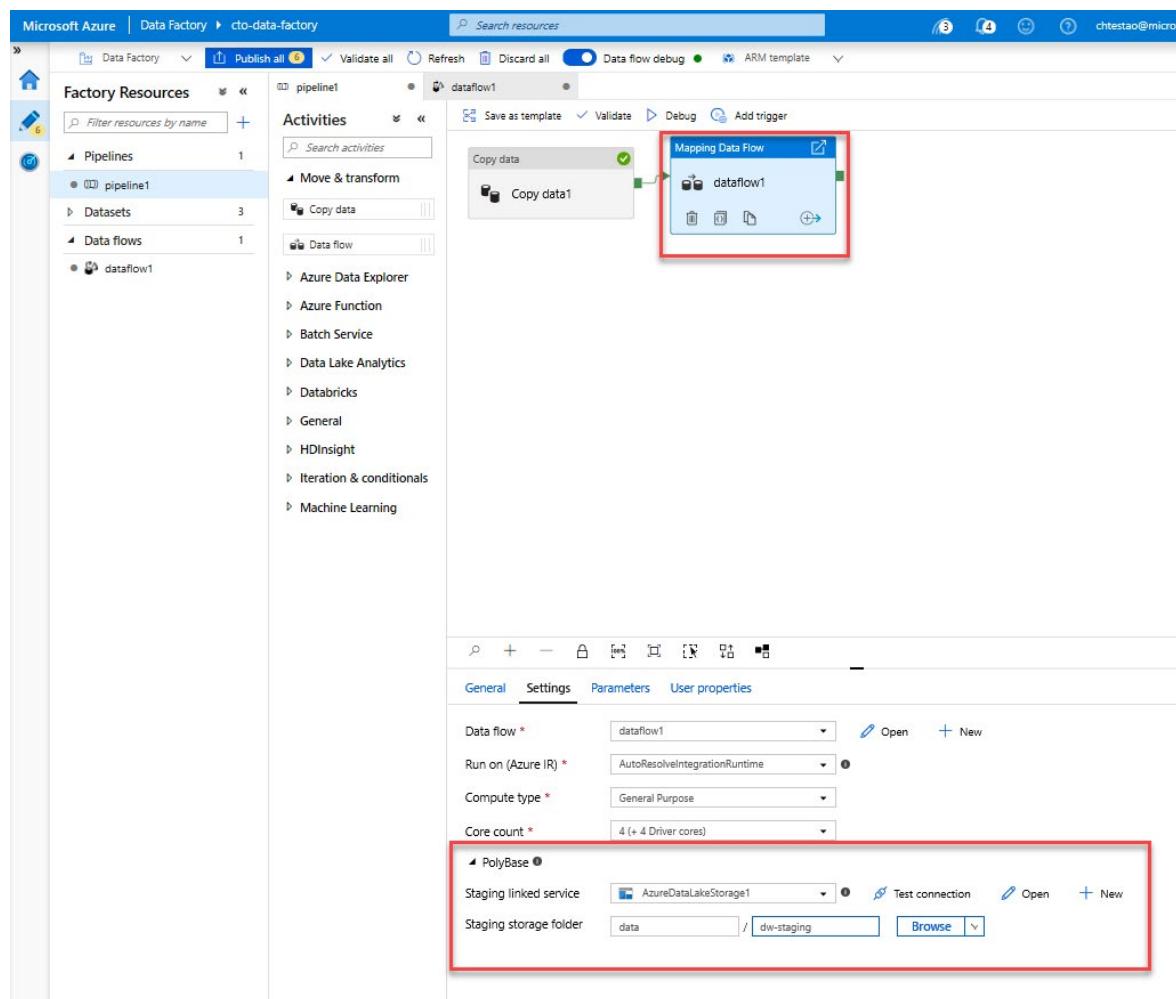


[./media/completed-mapping-data-flow.png)

Task 5: Running the Pipeline

- Go to the pipeline1 tab in the canvas. Because Azure Synapse Analytics in Data Flow uses **PolyBase**¹³, you must specify a blob or ADLS staging folder. In the Execute Data Flow activity's settings tab, open up the PolyBase accordion and select your ADLS linked service and specify a staging folder path.

¹³ <https://docs.microsoft.com/sql relational-databases/polybase/polybase-guide?view=sql-server-2017>



2. Before you publish your pipeline, run another debug run to confirm it's working as expected. Looking at the Output tab, you can monitor the status of both activities as they are running.
3. Once both activities succeeded, you can click on the eyeglasses icon next to the Data Flow activity to get a more in depth look at the Data Flow run.
4. If you used the same logic described in this lab, your Data Flow will write 737 rows to your SQL DW. You can go into **SQL Server Management Studio**¹⁴ to verify the pipeline worked correctly and see what got written.

¹⁴ <https://docs.microsoft.com/sql/ssms/download-sql-server-management-studio-ssms?view=sql-server-2017>

```

SQLQuery1.sql - dw..tosqladmin (2184)*  Object Explorer Details
Select count(*) as TotalCount from dbo.Ratings

Select * from dbo.Ratings
100 %  <  Results  Messages

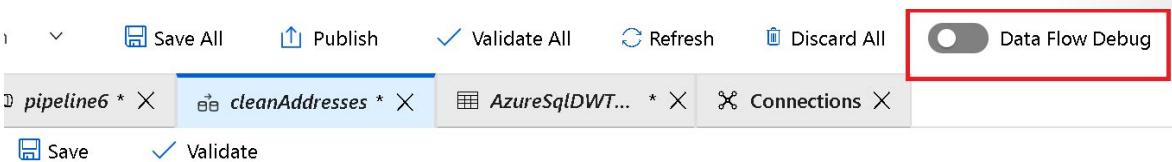
TotalCount
1  737

PrimaryGenre    year  AverageRating  HighestRated  LowestRated  NumberOfMovies
1  Action        1955  82.5          Dam Busters, The  To Hell and Back  4
2  (no genres listed)  1994  83           Freaky Friday   Freaky Friday   2
3  (no genres listed)  2012  63           Doctor Who: The Time of the Doctor  Doctor Who: The Time of the Doctor  2
4  Thriller      1963  51           Seven Days in May  Seven Days in May  2
5  (no genres listed)  2009  50           Boy Crazy       Boy Crazy       2
6  (no genres listed)  1964  89           Scorpio Rising  Scorpio Rising  2

```

Debug mapping data flow

During the building of Mapping Data Flows, you can interactively watch how the data transformations are executing so that you can debug them. To use this functionality, it is first necessary to turn on the "Data Flow Debug" feature.



Clicking Debug will provision the Spark clusters required to interact with the Mapping Data Flow transformations. On turning Debug on, you will be prompted to select the Integration Runtime that you require to use in the environment. If you select AutoResolveIntegrationRuntime, a cluster with eight cores that will be available with a time to live value of 60 minutes.

Note: It typically takes 5-7 minutes for the cluster to spin up. With this mode on and the Spark clusters running, you are able to build your data flow step by step and view the data as it runs through each transformation phase.

A Data Preview tab is available in Debug mode that will allow you to view the data at each stage of the pipeline. You can view the data after each transformation. The data previewer also provides the ability to actions on the data such as looking at descriptive statistics of the data, or the ability to modify the data.

movield	title	genres	year
1	Toy Story (1995)	Adventure Animation Children Comedy Fantasy	1995
2	Jumanji (1995)	Adventure Children Fantasy	1995
3	Grumpier Old Men (1995)	Comedy Romance	1995
4	Waiting to Exhale (1995)	Comedy Drama Romance	1995
5	Father of the Bride Part II (1995)	Comedy	1995
6	Heat (1995)	Action Crime Thriller	1995
7	Sabrina (1995)	Comedy Romance	1995
8	Tom and Huck (1995)	Adventure Children	1995
9	Sudden Death (1995)	Action	1995

Finally, you can use the debug settings to control the number of rows that are returned within the data previewer.

Note: It is recommended to limit the number of rows that returns enough to enable you to confirm that the data is correct. The bigger the data set, the longer it takes to return the results back. You can also use the Debug settings to specify any parameter values that should be used during the execution of the pipeline.

Use Azure Data Factory wrangling data

Wrangling Data Flow is a data flow object that can be added to the canvas designer as an activity in an Azure Data Factory pipeline to perform code free data preparation. It enables individuals who are not conversant with the traditional data preparation technologies such as Spark or SQL Server, and languages such as Python and T-SQL to prepare data at cloud scale iteratively.

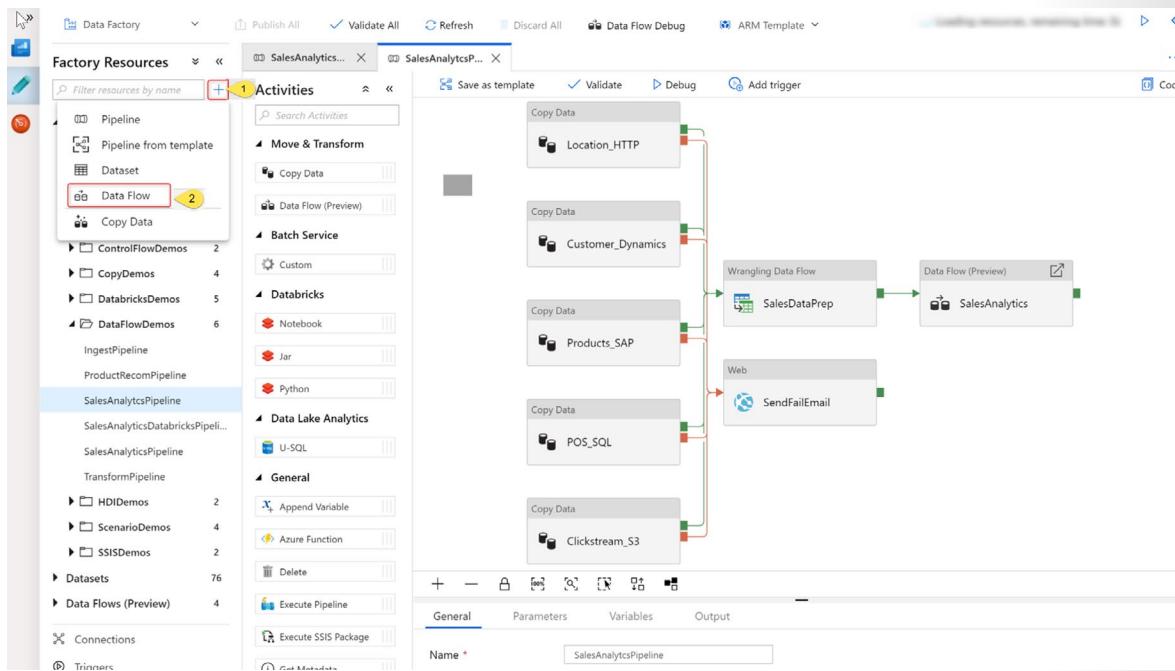
The Wrangling Data Flow uses a grid type interface for basic data preparation that is like the aesthetics of Excel, known as an Online Mashup Editor. The editor also enables more advanced users to perform more complex data preparation using formulas.

	A ^B _C EmployeeId	A ^B _C FirstName	A ^B _C LastName
1	1	"Harry"	"Potter"
2	2	"Hermione"	"Granger"
3	3	"Lord"	"Voldemort"
4	4	"Albus"	"Dumbledore"

The formulas work with Power Query Online and makes Power Query M functions available for data factory users. Wrangling data flow then translates the M language generated by the Power Query Online Mashup Editor into spark code for cloud scale execution.

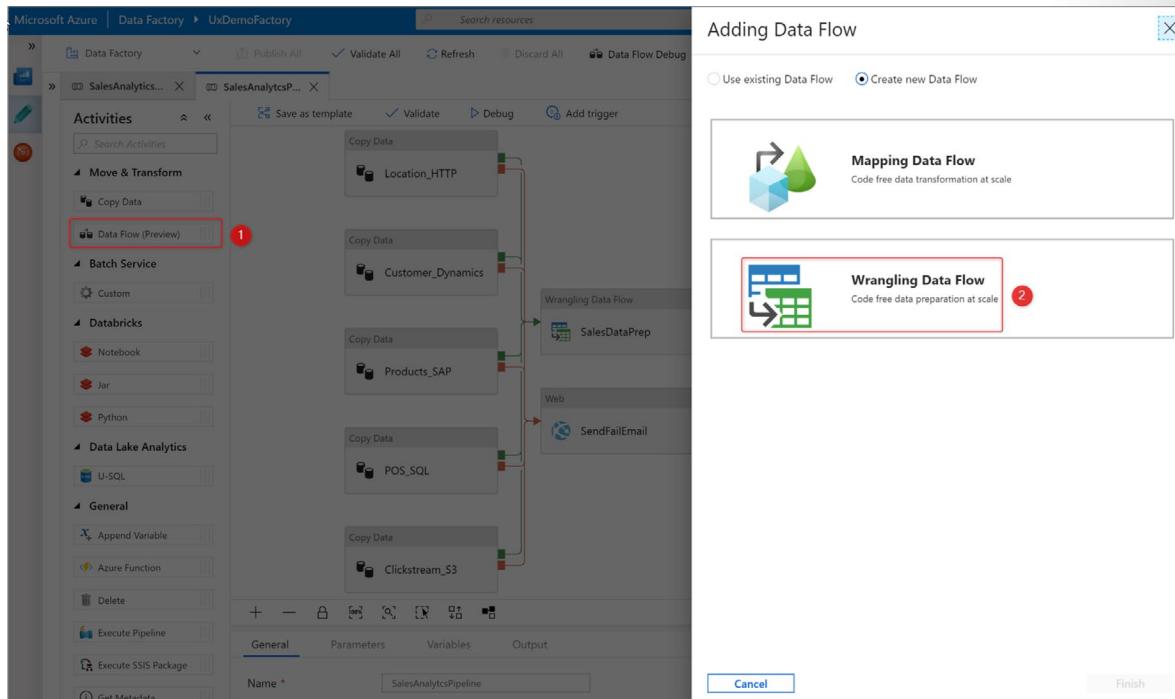
This capability enables both data engineers and citizen data integrators to interactively explore and prepare datasets. In addition, they can interactively work with the M language and preview the result before viewing it in the context of a wider pipeline.

There are two ways to create a wrangling data flow in Azure Data Factory. One way is to click the plus icon and select Data Flow in the factory resources pane.



The other method is in the activities pane of the pipeline canvas. Open the Move and Transform accordi-on and drag the Data flow activity onto the canvas.

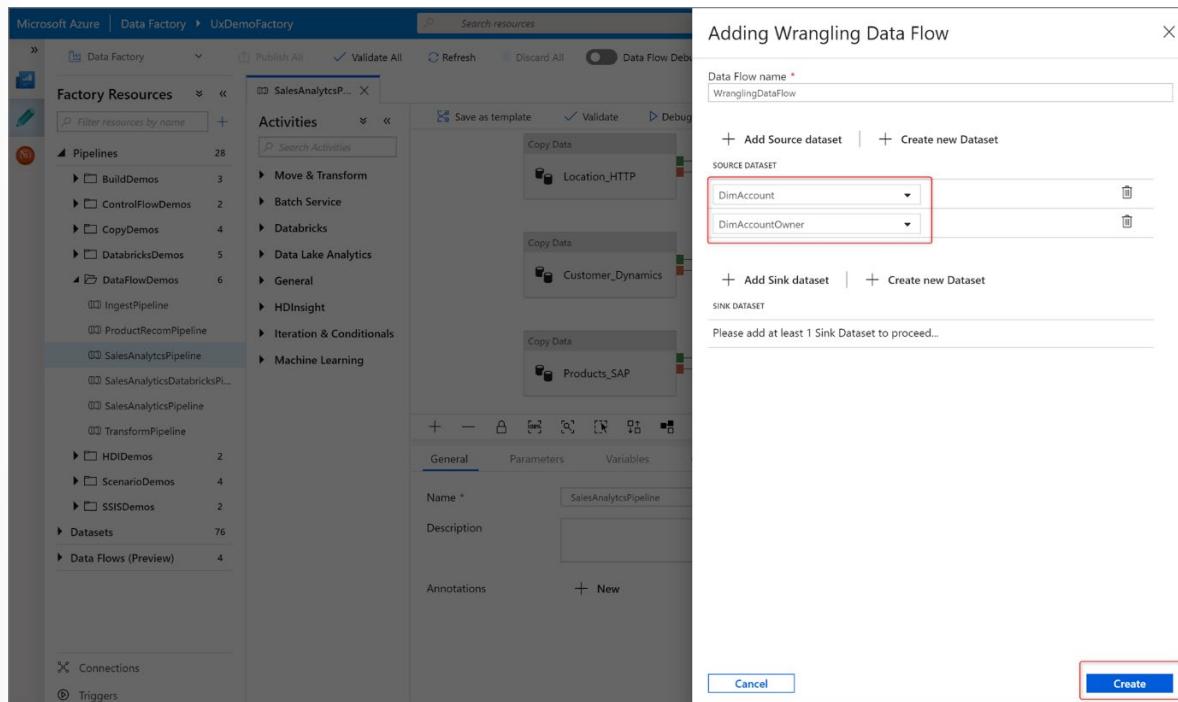
In both methods, in the side pane that opens, select Create new data flow and choose Wrangling data flow. Click OK.



Add a Source dataset for your wrangling data flow, and select a sink dataset. The following data sources are supported.

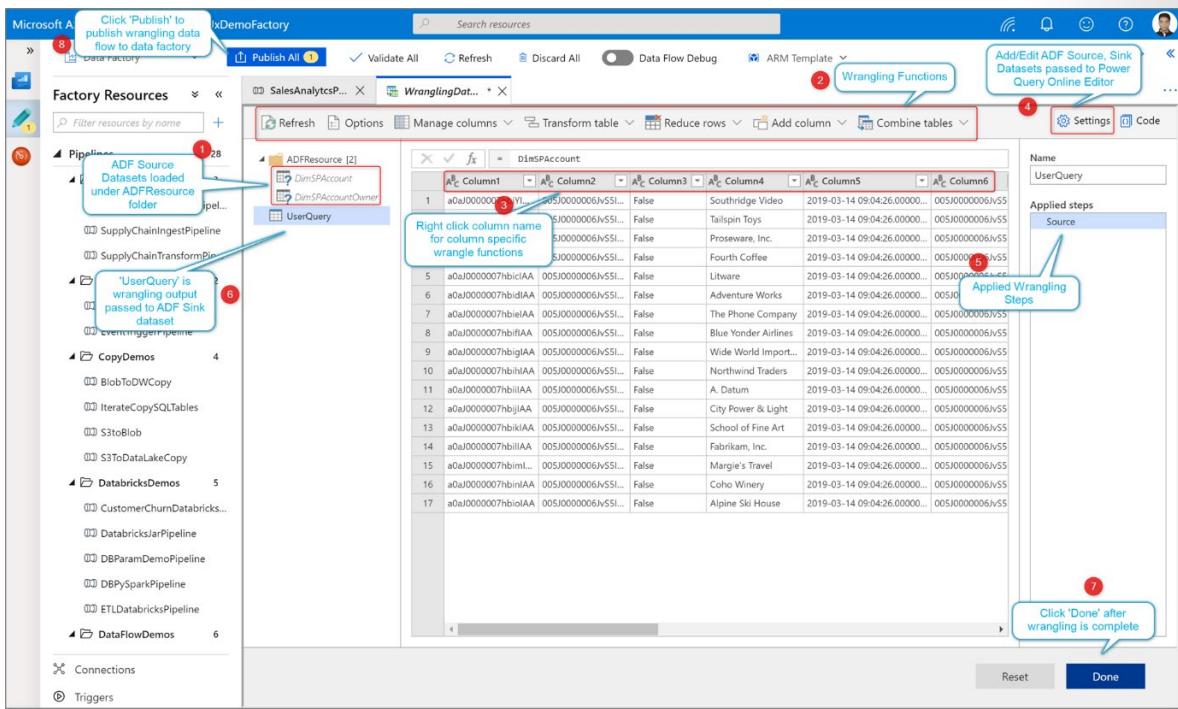
Connector	Data format	Authentication type
Azure Blob Storage	CSV, Parquet	Account Key
Azure Data Lake Storage Gen1	CSV	Service Principal
Azure Data Lake Storage Gen2	CSV, Parquet	Account Key, Service Principal
Azure SQL Database		SQL authentication
Azure Synapse Analytics		SQL authentication

Once you have selected a source, then click on create.



This opens the Online Mashup Editor.

navigating-wrangling-data-flow



It consists of the following components:

1. Dataset list.

This will provide the datasets that have been defined as the source for the Data Wrangling.

2. Wrangling Function toolbar.

The toolbar contains a variety of data wrangling functions that the user can access to manipulate the data including:

- Managing columns.
 - Transforming tables.
 - Reducing rows.
 - Adding columns.
 - Combining tables.

Each item is context-sensitive and contains sub functions specific to it.

3. Column headings.

As well as having the ability to rename columns, right-clicking the column will bring up context-sensitive items for managing columns.

4. Settings.

This enables you to add or edit data sources and data sinks, and modify setting for the wrangling data task.

5. Steps window.

This window shows the steps that have been applied to the wrangling output. In the example in the graphic, the step named "Source" has been applied the wrangling output named "UserQuery".

6. Wrangling output list.

Lists the data wrangling output that has been defined.

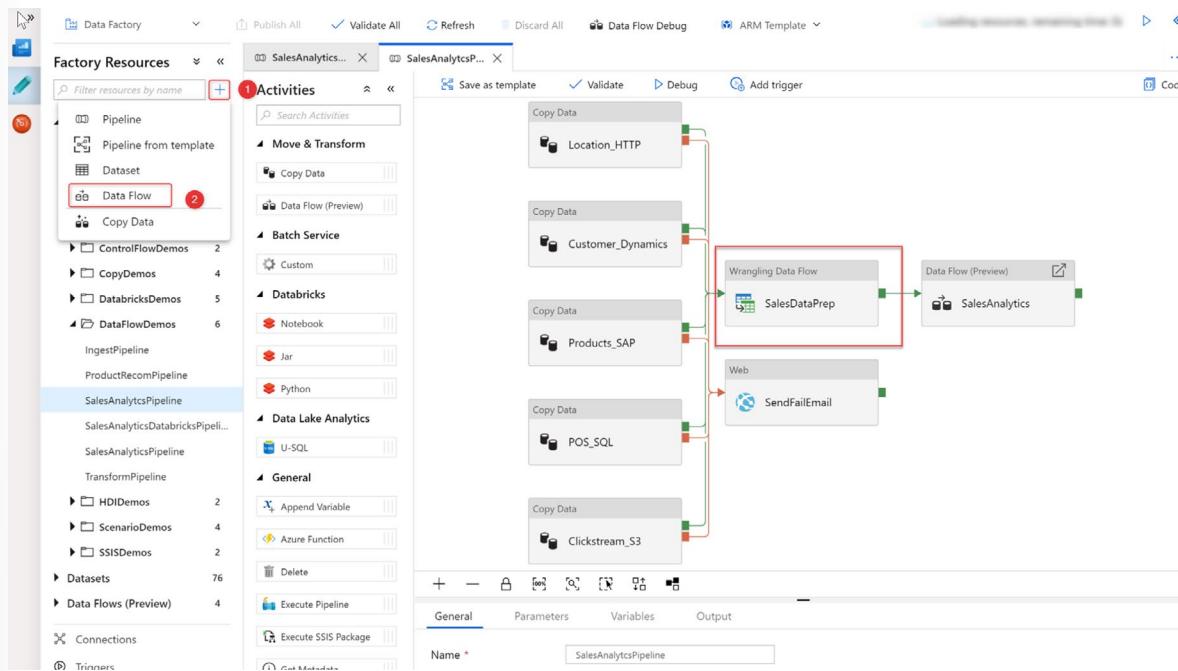
7. Done.

Completes the Data Wrangling task work.

8. Publish button.

Enables you to publish the work that has been created.

A wrangling data flow task appears in the canvass designer just like a Copy Activity task, or a Mapping Data Flow task and can be managed and monitored in the same way.



Use compute transformations within Azure Data Factory

In some cases, the code-free transformation at scale may not meet your requirements. You can use Azure Data Factory to ingest raw data collected from different sources and work with a range of compute resources such as Azure Databricks, Azure HDInsight, or other compute resources to restructure it as per your requirements.

ADF and Azure Databricks

As an example, the integration of Azure Databricks with ADF allows you to add Databricks notebooks within an ADF pipeline to leverage the analytical and data transformation capabilities of Databricks. You can add a notebook within your data workflow to structure and transform raw data loaded into ADF from different sources. Once the data is transformed using Databricks, you can then load it to any data warehouse source.

Data ingestion and transformation using the collective capabilities of ADF and Azure Databricks essentially involves the following steps:

- Create Azure storage account** - The first step is to create an Azure storage account to store your ingested and transformed data.

2. **Create an Azure Data Factory** - Once you have your storage account setup, you need to create your Azure Data Factory using Azure portal.
3. **Create data workflow pipeline** - After your storage and ADF is up and running, you start by creating a pipeline, where the first step is to copy data from your source using ADF's Copy activity. Copy Activity allows you to copy data from different on-premises and cloud sources.
4. **Add Databricks notebook to pipeline** - Once your data is copied to ADF, you add your Databricks notebook to the pipeline, after copy activity. This notebook may contain syntax and code to transform and clean raw data as required.
5. **Perform analysis on data** - Now that your data is cleaned up and structured into the required format, you can use Databricks notebooks to further train or analyze it to output required results.

You have learned what Azure Data Factory is and how its integration with Azure Databricks helps you to load and transform your data. Now let's create an end-to-end sample data workflow.

Integrating Azure Databricks notebooks with Azure Data Factory pipeline

There are a number of tasks that needs to be performed to integrate Azure Databricks notebooks with Azure Data Factory pipeline as follows:

1. Generate a Databricks Access Token.
2. Generate a Databricks Notebook
3. Create Linked Services
4. Create a Pipeline that uses Databricks Notebook Activity.
5. Trigger a Pipeline Run.

Note:The following steps assume there is already an Azure Databricks cluster already provisioned

Task 1: Generate a Databricks Access Token.

1. In the Azure portal, click on **Resource groups** and then click on **awrgstudxx**, and then click on **awdbwsstudxx** where xx are the initials of your name.
2. Click on **Launch Workspace**
3. Click the user **profile icon** in the upper right corner of your Databricks workspace.
4. Click **User Settings**.
5. Go to the Access Tokens tab, and click the **Generate New Token** button.
6. Enter a description in the **comment** "For ADF Integration" and set the **lifetime** period of 10 days and click on **Generate**
7. Copy the generated token and store in Notepad, and then click on **Done**.

Task 2: Generate a Databricks Notebook

1. On the left of the screen, click on the **Workspace** icon, then click on the arrow next to the word **Workspace**, and click on **Create** and then click on **Folder**. Name the folder **adftutorial**, and click on **Create Folder**. The adftutorial folder appears in the Workspace.
2. Click on the drop down arrow next to adftutorial, and then click **Create**, and then click **Notebook**.

3. In the Create Notebook dialog box, type the name of **mynotebook**, and ensure that the language states **Python**, and then click on **Create**. The notebook with the title of mynotebook appears/

4. In the newly created notebook "mynotebook" add the following code:

```
# Creating widgets for leveraging parameters, and printing the parameters
```

```
dbutils.widgets.text("input", "", "")  
dbutils.widgets.get("input")  
y = getArgument("input")  
print ("Param -\\'input\':")  
print (y)
```

Note:that the notebook path is **/adftutorial/mynotebook**

Task 3: Create Linked Services

1. In Microsoft Edge, click on the tab for the portal In the Azure portal, and return to Azure Data Factory.
2. In the **xx-data-factory** screen, click on **Author & Monitor**. Another tab opens up to author an Azure Data Factory solution.
3. On the left hand side of the screen, click on the **Author** icon. This opens up the Data Factory designer.
4. At the bottom of the screen, click on **Connections**, and then click on **+ New**.
5. In the **New Linked Service**, at the top of the screen, click on **Compute**, and then click on **Azure Databricks**, and then click on **Continue**.
6. In the **New Linked Service (Azure Databricks)** screen, fill in the following details and click on **Finish**
 - **Name:** xx_dbls, where xx are your initials
 - **Databricks Workspace:** awdbwsstudxx, where xx are your initials
 - **Select cluster:** use existing
 - **Domain/ Region:** should be populated
 - **Access Token:** Copy the access token from Notepad and paste into this field
 - **Choose from existing cluster:** awdbclstudxx, where xx are your initials
 - Leave other options to their default settings

Note:When you click on finish, you are returned to the **Author & Monitor** screen where the xx_dbls has been created, with the other linked services created in the previous exercise.

Task 4: Create a pipeline that uses Databricks Notebook Activity.

1. On the left hand side of the screen, under Factory Resources, click on the + icon, and then click on **Pipeline**. This opens up a tab with a Pipeline designer.
2. At the bottom of the pipeline designer, click on the parameters tab, and then click on **+ New**
3. Create a parameter with the Name of **name**, with a type of **string**
4. Under the **Activities** menu, expand out **Databricks**.
5. Click and drag **Notebook** onto the canvas.

6. In the properties for the **Notebook1** window at the bottom, complete the following steps:
 - Switch to the **Azure Databricks** tab.
 - Select **xx_db1s** which you created in the previous procedure.
 - Switch to the **Settings** tab, and put **/adftutorial/mynotebook** in Notebook path.
 - Expand **Base Parameters**, and then click on **+ New**
 - Create a parameter with the Name of **input**, with a value of **@pipeline().parameters.name**
7. In the **Notebook1**, click on **Validate**, next to the Save as template button. A window appears on the right of the screen that states "Your Pipeline has been validated. No errors were found." Click on the **>>** to close the window.
8. Click on the **Publish All** to publish the linked service and pipeline.
Note: A message will appear to state that the deployment is successful.

Task 5: Trigger a Pipeline Run

1. In the **Notebook1**, click on **Add trigger**, and click on **Trigger Now** next to the **Debug** button.
2. The **Pipeline Run** dialog box asks for the name parameter. Use **/path/filename** as the parameter here. Click **Finish**. A red circle appears above the Notebook1 activity in the canvas.

Task 6: Monitor the Pipeline

1. On the left of the screen, click on the **Monitor** tab. Confirm that you see a pipeline run. It takes approximately 5-8 minutes to create a Databricks job cluster, where the notebook is executed.
2. Select **Refresh** periodically to check the status of the pipeline run.
3. To see activity runs associated with the pipeline run, select **View Activity Runs** in the **Actions** column.

Task 7: Verify the output

1. In Microsoft Edge, click on the tab **mynotebook - Databricks**
2. In the **Azure Databricks** workspace, click on **Clusters** and you can see the Job status as pending execution, running, or terminated.
3. Click on the cluster **awdbclstudxx**, and then click on the **Event Log** to view the activities.

Note: You should see an Event Type of **Starting** with the time you triggered the pipeline run.

Integrate SQL server integration services packages within Azure Data Factory

You may work in an organization where much of the transformation logic is currently held in existing SSIS packages that have been created on SQL Server. You have the ability to lift and shift SSIS package so you can execute them within Azure Data Factory, so you can make use in existing work. In order to do this you must set up an Azure-SSIS integration runtime.

Azure-SSIS integration runtime

In order to make use of the Azure-SSIS integration runtime, it is assumed that there is SSIS Catalog (SSISDB) deployed on a SQL Server SSIS instance. With that prerequisite met, the Azure-SSIS integration runtime is capable of:

- Lift and shift existing SSIS workloads

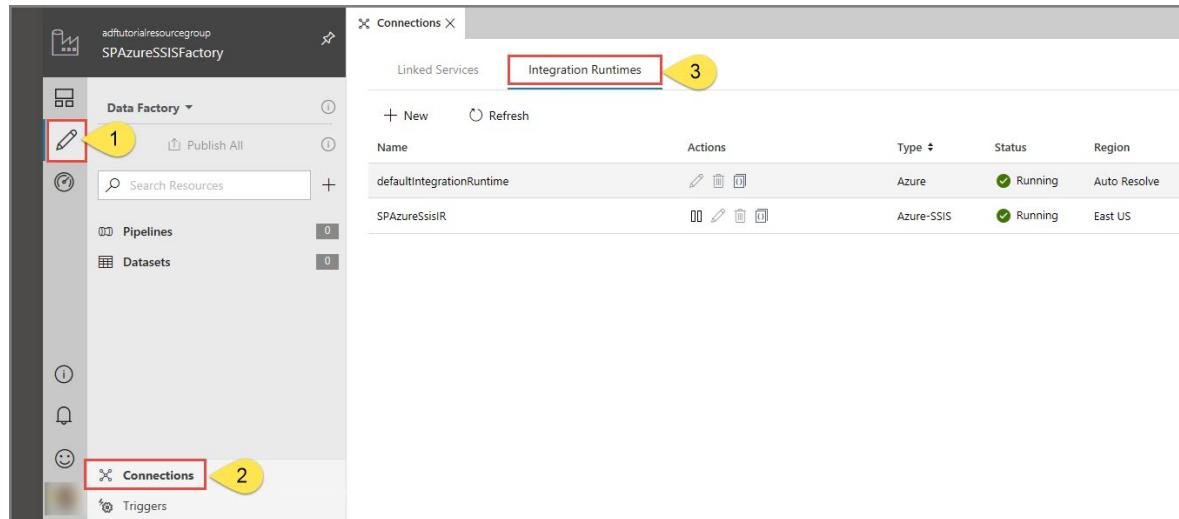
During the provisioning of the Azure-SSIS integration runtime, you specify the following options:

- The node size (including the number of cores) and the number of nodes in the cluster.
- The existing instance of Azure SQL Database to host the SSIS Catalog Database (SSISDB), and the service tier for the database.
- The maximum parallel executions per node.

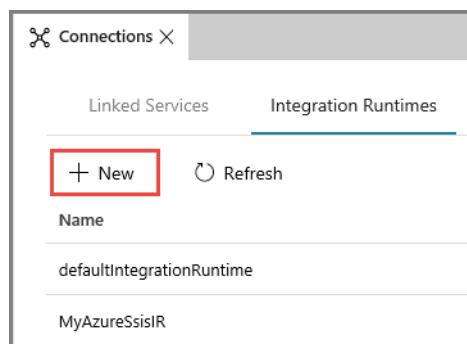
With the Azure-SSIS integration runtime enabled, you are able to manage, monitor and schedule SSIS packages using tools such as SQL Server Management Studio (SSMS) or SQL Server Data Tools (SSDT).

Create an Azure-SSIS integration runtime

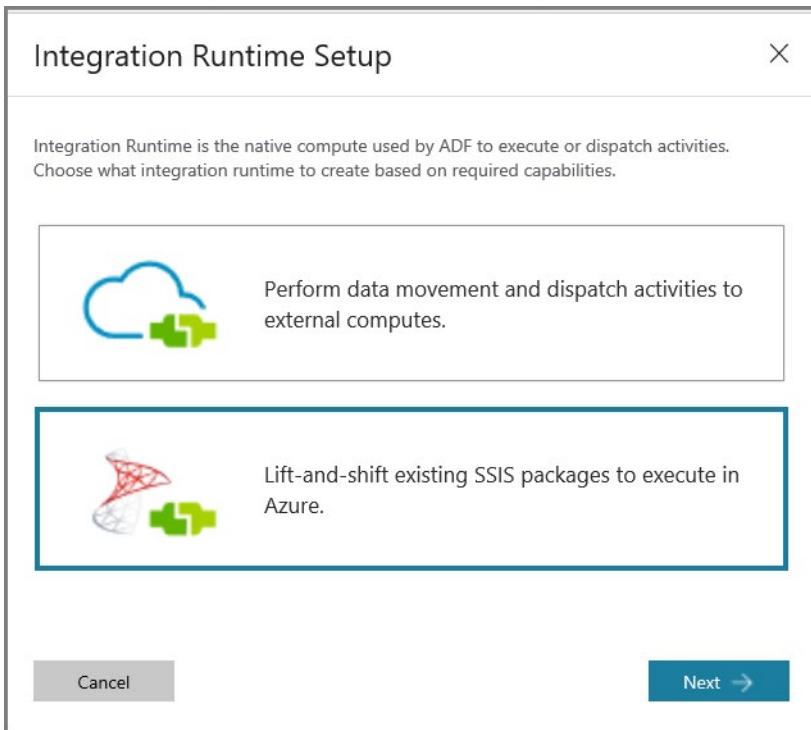
1. In the Azure Data Factory designer , in the **Edit** tab, click **Connections**. Click on the **Integration Runtimes** tab to view existing integration runtimes in your data factory.



2. Click **+ New** to create an Azure-SSIS IR and open the **Integration runtime setup** pane.



3. In the **Integration runtime setup** pane, select the **Lift-and-shift existing SSIS packages to execute in Azure** tile, and then select **Next**.



4. On selecting this option, there are three types of settings to configure

General settings page

1. On the **General settings** page of **Integration runtime setup** pane, complete the following steps.

Integration Runtime Setup X

General Settings

Name * (i)
integrationRuntime1

Description (i)

Type (i)
Azure-SSIS

Location * (i)
West Europe ▼

Node Size * (i)
Standard_D4_v2 (8 Core(s), 28672 MB) ▼

Node Number * (i)
 2

Edition/License * (i)
Standard ▼

Save Money

Save with a license you already own. Already have a SQL Server license? Yes No

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

Cancel Next →

2. In **Name**, enter the name of your integration runtime.
3. For **Description**, enter the description of your integration runtime.
4. For **Location**, select the location of your integration runtime. It is recommended that you select the same location of your database server to host SSISDB.
5. For **Node Size**, select the size of node in your integration runtime cluster.
6. For **Node Number**, select the number of nodes in your integration runtime cluster.
7. For **Edition/License**, select the SQL Server edition for your integration runtime.
8. For **Save Money**, select the Azure Hybrid Benefit option for your integration runtime: Select **Yes** if you want to bring your own SQL Server license with Software Assurance to benefit from cost savings with hybrid use.
9. Select **Next**.

Deployment settings page

1. On the **Deployment settings** page of **Integration runtime setup** pane, complete the following steps.
2. Click the **Create SSIS catalog (SSISDB) hosted by Azure SQL Database server/Managed Instance to store your projects/packages/environments/execution logs** check box to choose the package deployment mode.

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 *

Location

Catalog database server endpoint *

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

Admin username *

Admin password *

Catalog database service tier *

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

3. For **Subscription**, select the Azure subscription that has your database server to host SSISDB.
4. For **Location**, select the location of your database server to host SSISDB. We recommend that you select the same location of your integration runtime.
5. For **Catalog Database Server Endpoint**, select the endpoint of your database server to host SSISDB.
6. Select the **Use Azure Active Directory (Azure AD) authentication with the managed identity for your ADF** check box to choose the authentication method for your database server to host SSISDB.

7. For **Admin Username**, enter the SQL authentication username for your database server to host SSISDB.
8. For **Admin Password**, enter the SQL authentication password for your database server to host SSISDB.
9. For **Catalog Database Service Tier**, select the service tier for your database server to host SSISDB. Select the Basic, Standard, or Premium tier, or select an elastic pool name.

The alternative approach is to:

1. Select the **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** check box to choose whether you want to manage your packages that are deployed into MSDB, file system, or Azure Files (Package Deployment Model) with Azure-SSIS IR package stores.

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](#))

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

[New](#) | [Delete](#)

<input type="checkbox"/>	NAME	TYPE
	myAzureFilesPackageStore	Azure Files
	mySQLMIPackageStore	Azure SQL Database Managed Instance

[Continue](#) [Back](#) [Cancel](#)

2. On the **Add package store** pane, complete the following steps.
3. For **Package store name**, enter the name of your package store.

4. For **Package store linked service**, select your existing linked service that stores the access information for file system/Azure Files/Azure SQL Managed Instance where your packages are deployed or create a new one by selecting **New**. On the **New linked service** pane, complete the following steps.

New linked service

Name *****
linkedService1

Description

Type *****
Azure File Storage

Connect via integration runtime *****
AutoResolveIntegrationRuntime

Account selection method
 From Azure subscription Enter manually

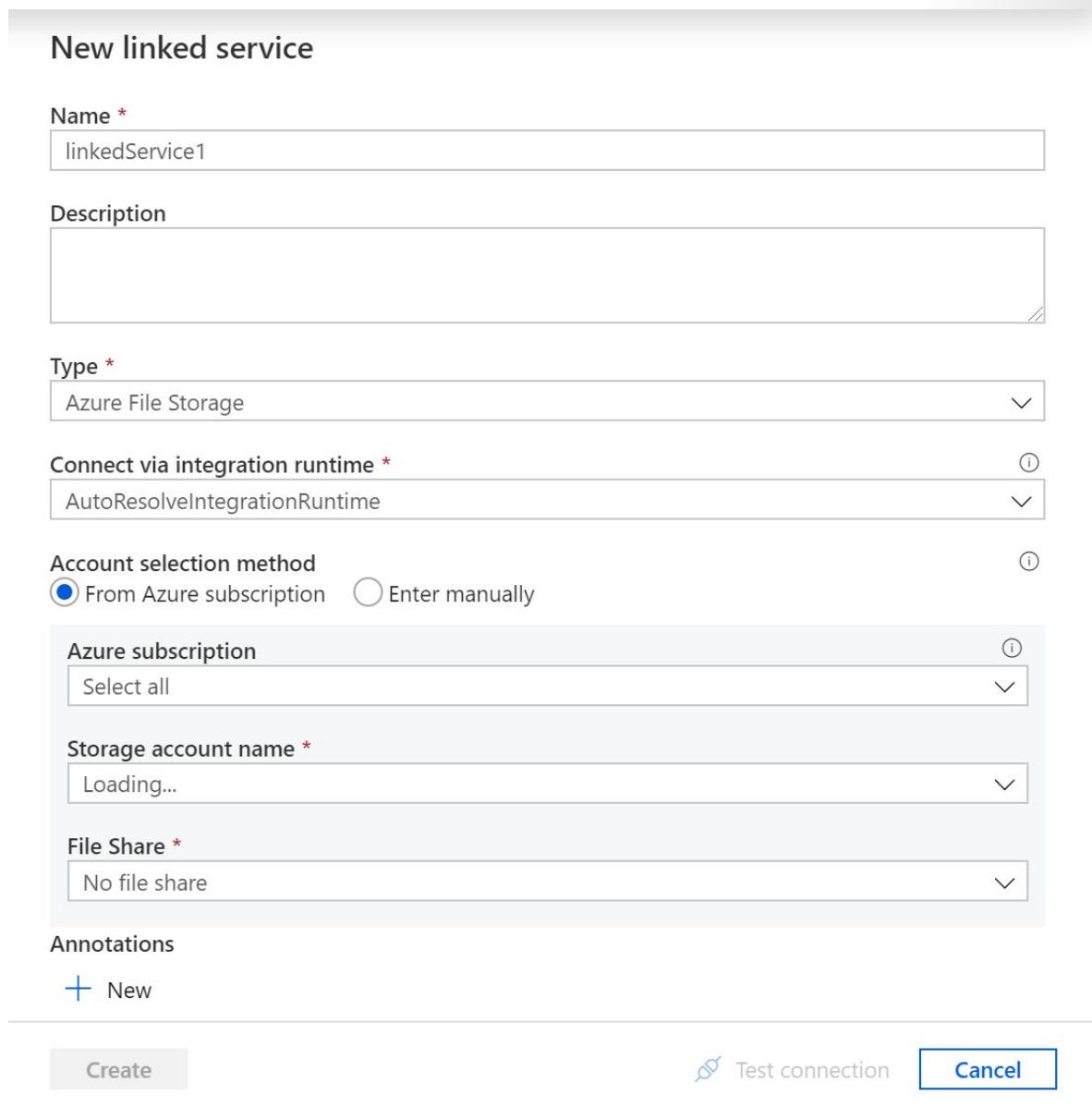
Azure subscription
Select all

Storage account name *****
Loading...

File Share *****
No file share

Annotations
+ New

Create Test connection Cancel



5. For **Name**, enter the name of your linked service.
6. For **Description**, enter the description of your linked service.
7. For **Type**, select **Azure File Storage**, **Azure SQL Managed Instance**, or **File System**.
8. You can ignore **Connect via integration runtime**, since we always use your Azure-SSIS IR to fetch the access information for package stores.
9. If you select **Azure File Storage**, complete the following steps.
10. For **Account selection method**, select **From Azure subscription** or **Enter manually**.
11. If you select **From Azure subscription**, select the relevant **Azure subscription**, **Storage account name**, and **File share**.

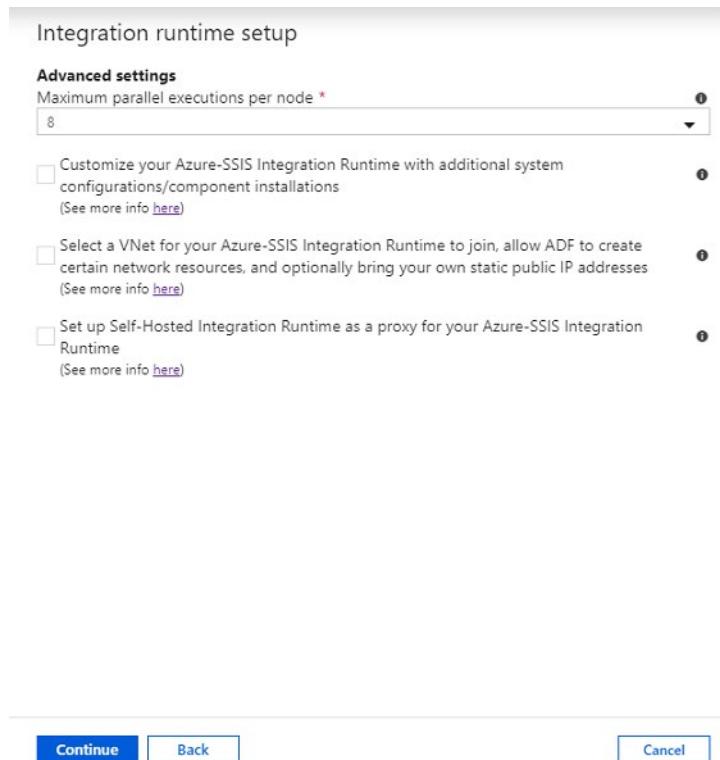
12. If you select **Enter manually**, enter \\<storage account name>.file.core.windows.net\<- file share name> for **Host**, Azure\<storage account name> for **Username**, and <storage account key> for **Password** or select your **Azure Key Vault** where it's stored as a secret.

NOTE: There are different settings if you select **Azure SQL Managed Instance**, or **File System**

13. Select **Test connection** when applicable and if it's successful, select **Next**.

Advanced settings page

1. On the **Advanced settings** page of **Integration runtime setup** pane, complete the following steps.



2. For **Maximum Parallel Executions Per Node**, select the maximum number of packages to run concurrently per node in your integration runtime cluster.
3. Select the **Customize your Azure-SSIS Integration Runtime with additional system configurations/component installations** check box to choose whether you want to add standard/express custom setups on your Azure-SSIS IR.
4. Select the **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** check box to choose whether you want to join your Azure-SSIS IR to a virtual network.
5. Select the **Set up Self-Hosted Integration Runtime as a proxy for your Azure-SSIS Integration Runtime** check box to choose whether you want to configure a self-hosted IR as proxy for your Azure-SSIS IR. For more information.
6. Click **Continue**.
7. On the **Summary**, review all provisioning settings, and select **Finish** to start the creation of your integration runtime.

8. On the **Connections** pane of **Manage** hub, switch to the **Integration runtimes** page and select **Refresh**.

NAME ↑↓	TYPE ↑↓	SUB-TYPE ↑↓	STATUS ↑↓	REGION ↑↓
AutoResolveIntegrationRuntime	Azure	Public	Running	Auto Resolve
myAzureSSISIntegrationRuntime	Azure-SSIS	---	Stopped	East US

Knowledge check

Question 1

Which transformation in the Mapping Data Flow is used to routes data rows to different streams based on matching conditions?

- Lookup.
- Conditional Split.
- Select.

Question 2

Which transformation is used to load data into a data store or compute resource?

- Window.
- Source.
- Sink.

Summary

The Mapping Data Flow makes it even easier for more people to create code free transformation that can be used to cleanse data. With common transformation types covered, it may be the only tool required before loading into a data warehouse or preparing for an advanced analytical workload. If not, Azure Data Factory also can call on the features of a variety of Azure Data Platform technologies to perform transformations on its behalf. For organizations that have SSIS packages that contain the transformation logic, SSIS packages can be used with Azure Data Factory too, which make Azure Data Factory the one stop shop for your transformation needs.

Important: Should you be working in your own subscription while running through this content, it is best practice at the end of a project to identify whether or not you still need the resources you created.

Resources left running can cost you money. You can delete resources one by one, or just delete the resource group to get rid of the entire set.

Module Lab information

Lab 6 - Transform data with Azure Data Factory or Azure Synapse Pipelines

- **Estimated Time:** 50 - 60 minutes
- **Lab files:** The files for this lab are located in the *Instructions\Labs\08* folder.

Lab overview

This lab teaches students how to build data integration pipelines to ingest from multiple data sources, transform data using mapping data flows and notebooks, and perform data movement into one or more data sinks.

Lab objectives

After completing this lab, you will be able to:

- Execute code-free transformations at scale with Azure Synapse Pipelines
- Create data pipeline to import poorly formatted CSV files
- Create Mapping Data Flows

Module summary

module-summary

In this module, you have learned how to perform data integration with Azure Data Factory or Azure Synapse pipelines. Students understand how to execute code-free transformation at scale with Azure Data Factory and Azure Synapse pipelines.

Learning objectives

In this module, you have learned to:

- Perform Data integration with Azure Data Factory or Azure Synapse Pipelines
- Perform Code-free transformation at scale with Azure Data Factory or Azure Synapse Pipelines

Post Course Review

After the course, consider visiting **the Microsoft Customer Case Study site¹⁵**. Use the search bar to search by an industry such as healthcare or retail, or by a technology such as Azure Synapse Analytics or Azure Databricks. Read through some of the customers stories.

Important

Remember

Should you be working in your own subscription while running through this content, it is best practice at the end of a project to identify whether or not you still need the resources you created. Resources left running can cost you money.

You can delete resources one by one, or just delete the resource group to get rid of the entire set.

¹⁵ <https://customers.microsoft.com/>

Answers

Question 1

Which Azure Data Factory component orchestrates a transformation job or runs a data movement command?

- Linked Services
- Datasets
- Activities

Explanation

Correct. Activities contains the transformation logic or the analysis commands of the Azure Data Factory's work.

Question 2

You are moving data from an Azure Data Lake Gen2 store to Azure Synapse Analytics. Which Azure Data Factory integration runtime would be used in a data copy activity?

- Azure-SSIS
- Azure
- Self-hosted

Explanation

Correct. When moving data between Azure data platform technologies, the Azure Integration runtime is used when copying data between two Azure data platform.

Question 1

Which transformation in the Mapping Data Flow is used to routes data rows to different streams based on matching conditions?

- Lookup.
- Conditional Split.
- Select.

Explanation

Correct. A Conditional Split transformation routes data rows to different streams based on matching conditions. The conditional split transformation is similar to a CASE decision structure in a programming language.

Question 2

Which transformation is used to load data into a data store or compute resource?

- Window.
- Source.
- Sink.

Explanation

Correct. A Sink transformation allows you to choose a dataset definition for the destination output data. You can have as many sink transformations as your data flow requires.

Module 7 Integrate Data from Notebooks with Azure Data Factory or Azure Synapse Pipelines

Module introduction

module-introduction

In this module, students will learn how to orchestrate data movement and transformations in Azure Data Factory or Azure Synapse Pipeline. Students will learn how Azure Data Factory or Azure Synapse pipelines can orchestrate large scale data movement by using other Azure Data Platform and Machine Learning technologies.

Learning objectives

In this module, you will:

- Understand how to add an activity to the control flow to orchestrate data from other technologies
- Understand how to use parameters in Azure Data Factory/Synapse pipelines

Orchestrating data movement and transformation in Azure Data Factory

Introduction

Learn how to orchestrate data movement and transformations in Azure Data Factory.

After completing this module, you will be able to:

- Understand the data factory control flow
- Work with data factory pipelines
- Debug data factory pipelines
- Add parameters to data factory components
- Execute data factory packages

Prerequisites

Before taking this module, it is recommended that the student is able to:

- Log into the Azure portal
- Explain and create resource groups
- Describe Azure Data Factory and its core components
- Ingest data into Azure Data Factory using the Copy Activity

Understand data factory control flow

What is control flow

Control flow is an orchestration of pipeline activities that includes chaining activities in a sequence, branching, defining parameters at the pipeline level, and passing arguments while invoking the pipeline on demand or from a trigger.

Control flow can also include looping containers, that can pass information for each iteration of the looping container.

If a For Each loop is used as a control flow activity, Azure Data Factory can start multiple activities in parallel using this approach. This allows you to build complex and iterative processing logic within the pipelines you create with Azure Data Factory, which supports the creation of diverse data integration patterns such as building a modern data warehouse.

Some of the common control flow activities are described in the below sections.

Chaining activities

Within Azure Data Factory you can chain activities in a sequence within a pipeline.

It is possible to use the **dependsOn** property in an activity definition to chain it with an upstream activity.

Branching activities

Use Azure Data Factory for branching activities within a pipeline. An example of a branching activity is *The If-condition* activity which is similar to an if-statement provided in programming languages. A branching activity evaluates a set of activities, and when the condition evaluates to true, a set of activities are executed. When it evaluates to false, then an alternative set of activities is executed.

Parameters

You can define parameters at the pipeline level and pass arguments while you're invoking the pipeline on-demand or from a trigger. Activities then consume the arguments held in a parameter as they are passed to the pipeline.

Custom state passing

Custom state passing is made possible with Azure Data Factory. Custom state passing is an activity that creates output or the state of the activity that needs to be consumed by a subsequent activity in the pipeline. An example is that in a JSON definition of an activity, you can access the output of the previous activity. Using custom state passing enables you to build workflows where values are passing through activities.

Looping containers

The looping containers umbrella of control flow such as the ForEach activity defines repetition in a pipeline.

It enables you to iterate over a collection and runs specified activities in the defined loop. It works similarly to the 'for each looping structure' used in programming languages.

Besides each activity, there is also an Until activity. This functionality is similar to a do-until loop used in programming. What it does is running a set of activities (do) in a loop until the condition (until) is met.

Trigger based flows

Pipelines can be triggered by on-demand (event-based, i.e. blob post) or wall-clock time.

Invoke a pipeline from another pipeline

The Execute Pipeline activity with Azure Data Factory allows a Data Factory pipeline to invoke another pipeline.

Delta flows

Use-cases related to using delta flows are delta loads.

Delta loads in ETL patterns will only load data that has changed since a previous iteration of a pipeline. Capabilities such as lookup activity, and flexible scheduling helps handling delta load jobs.

In the case of using a Lookup activity, it will read or look up a record or table name value from any external source. This output can further be referenced by succeeding activities.

Other control flows

There are many more control flow activities. Below you find a couple of other useful activities.

- Web activity:
The web activity in Azure Data Factory using control flows, can call a custom REST endpoint from a Data Factory pipeline.
Datasets and linked services can be passed in order to get consumed by the activity.
- Get metadata activity:
The Get metadata activity retrieves the metadata of any data in Azure Data Factory.

Work with data factory pipelines

In order to work with data factory pipelines, it is imperative to understand what a pipeline in Azure Data Factory is.

A pipeline in Azure Data Factory represents a logical grouping of activities where the activities together perform a certain task.

An example of a combination of activities in one pipeline can be, ingesting and cleaning log data in combination with a mapping data flow that analyzes the log data that has been cleaned.

A pipeline enables you to manage the separate individual activities as a set, which would otherwise be managed individually.

It enables you to deploy and schedule the activities efficiently, through the use of a single pipeline, versus managing each activity independently.

Activities in a pipeline are referred to as actions that you perform on your data.

An activity can take zero or more input datasets and produce one or more output datasets.

An example of an action can be the use of a copy activity, where you copy data from an Azure SQL Database to an Azure DataLake Storage Gen2.

To build on this example, you can use a data flow activity or an Azure Databricks Notebook activity for processing and transforming the data that was copied to your Azure Data Lake Storage Gen2 account, in order to have the data ready for business intelligence reporting solutions like in Azure Synapse Analytics.

Since there are many activities that are possible in a pipeline in Azure Data Factory, we have grouped the activities in three categories:

- *Data movement activities*: the Copy Activity in Data Factory copies data from a source data store to a sink data store.
- *Data transformation activities*: Azure Data Factory supports transformation activities such as Data Flow, Azure Function, Spark, and others that can be added to pipelines either individually or chained with another activity.
- *Control activities*: Examples of control flow activities are 'get metadata', 'For Each', and 'Execute Pipeline'.

Activities can depend on each other.

What we mean, is that the activity dependency defines how subsequent activities depend on previous activities.

The dependency itself can be based on a condition of whether to continue in the execution of previous defined activities in order to complete a task. An activity that depends on one or more previous activities, can have different dependency conditions.

The four dependency conditions are:

- Succeeded
- Failed
- Skipped
- Completed

For example, if a pipeline has an Activity A, followed by an Activity B and Activity B has as a dependency condition on Activity A 'Succeeded', then Activity B will only run if Activity A has the status of succeeded.

If you have multiple activities in a pipeline and subsequent activities are not dependent on previous activities, the activities may run in parallel.

Debug data factory pipelines

Customer requirements and expectations are changing in relation to data integration.

The need among users to develop and debug their Extract Transform/Load (ETL) and Extract Load/Transform (ELT) workflows iteratively is therefore becoming more imperative.

Azure Data Factory can help you build and develop iterative debug Data Factory pipelines when you develop your data integration solution.

By authoring a pipeline using the pipeline canvas, you can test your activities and pipelines by using the Debug capability.

In Azure Data Factory, there is no need to publish changes in the pipeline or activities before you want to debug.

This is helpful in a scenario where you want to test the changes and see if it works as expected before you actually save and publish them.

Sometimes, you don't want to debug the whole pipeline but test a part of the pipeline. A Debug run allows you to do just that.

You can test the pipeline end to end or set a breakpoint.

By doing so in debug mode, you can interactively see the results of each step while you build and debug your pipeline.

Debug and publish a pipeline:

As you create or modify a pipeline that is running, you can see the results of each activity in the Output tab of the pipeline canvas.

After a test run succeeds, and you are satisfied with the results, you can add more activities to the pipeline and continue debugging in an iterative manner.

When you are not satisfied or like to stop the pipeline from debugging, you can cancel a test run while it is in progress.

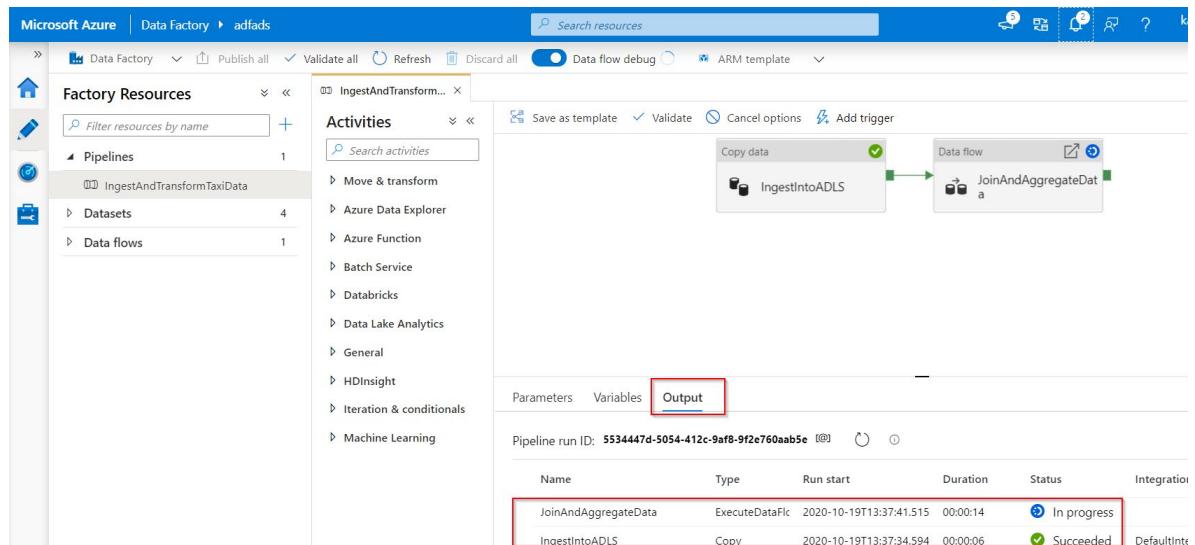
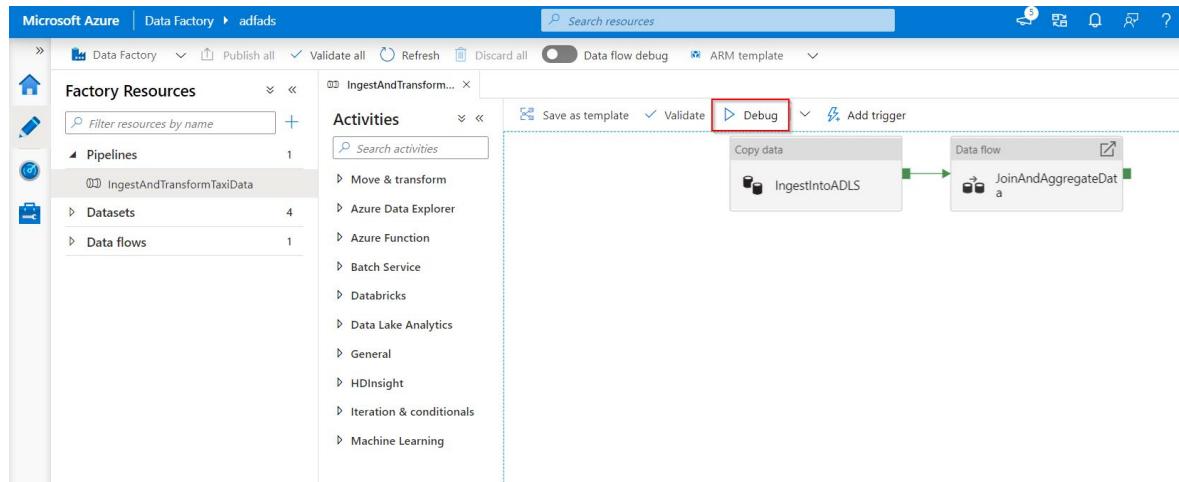
You do need to be aware that by selecting the debug slider, it will actually run the pipeline.

Therefore, if the pipeline contains, for example, a copy activity, the test run will copy data from source to destination.

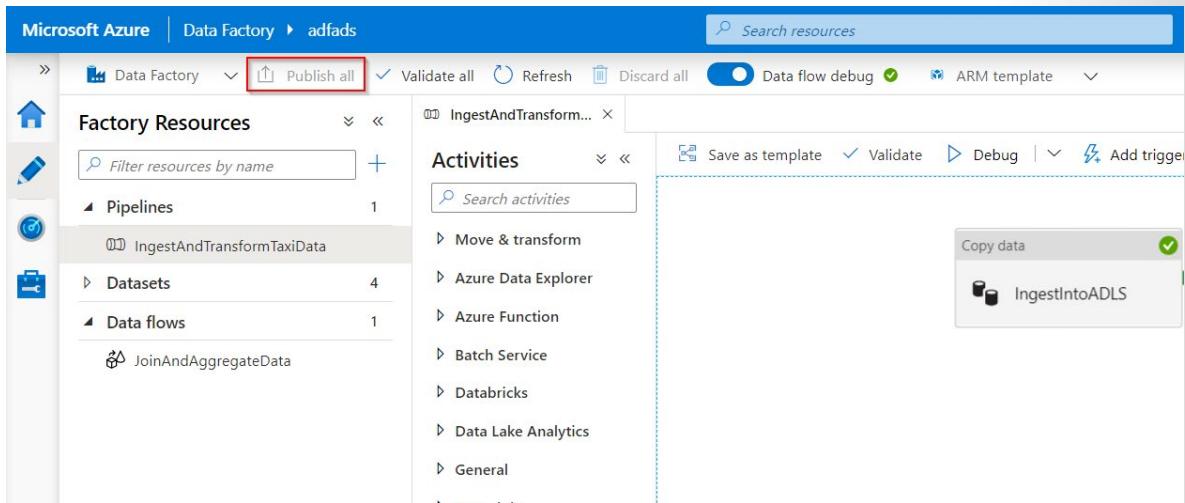
A best practice is to use test folders in your copy activities and other activities when debugging such that

when you are satisfied with the results and have debugged the pipeline, you switch to the actual folders for your normal operations.

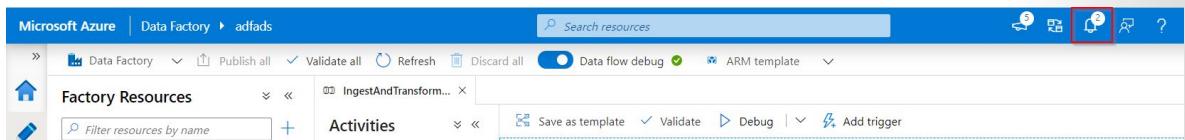
1. To debug the pipeline, select Debug on the toolbar. You see the status of the pipeline run in the Output tab at the bottom of the window.



2. Once the pipeline can run successfully, in the top toolbar, select Publish all. This action publishes entities (datasets, and pipelines) you created to Data Factory.



- Wait until you see the successfully published message. To see notification messages, click the Show Notifications on the top-right (bell button).



Mapping dataflow debug:

During the building of Mapping Data Flows, you can interactively watch how the data shapes and transformations are executing so that you can debug them. To use this functionality, it is first necessary to turn on the "Data Flow Debug" feature.

The debug session can be used both in Data Flow design sessions as well as during pipeline debug execution of data flows.

Once the debug mode is on, you will actually build the data flow with an active Spark Cluster. The Spark cluster will close once the debug is off.

You do have a choice in what compute you're going to use. When you use an existing debug cluster, it will reduce the start-up time.

However, for complex or parallel workloads you might want to spin up your own just-in-time cluster.

Best practices for debugging data flows, is to keep the debug mode on, to check and validate the business logic included in the data flow.

Visually viewing the data transformations and shapes helps you see the changes.

If you want to test the dataflow in a pipeline that you've created it is best to use "Debug" button on the pipeline panel.

While data preview doesn't write data, a debug run within your dataflow will write, just like debugging a pipeline, data to your sink destination.

Debug settings

As mentioned, each debug session that is started from the Azure Data Factory User Interface, is considered a new session with its own Spark cluster.

In order to monitor the sessions, you can use the monitoring view for debug session to manage debug

sessions per Data Factory that has been set up.

To see whether a Spark cluster is ready for debugging, you can check the cluster status indication at the top of the design surface. If it's green it's ready, if the cluster wasn't running yet when you entered debug mode, the waiting time could be around 5-7 minutes since the clusters need to spin up.

It is best practice that once you finish debugging that you switch debug mode off such that the Spark cluster terminates.

When you're debugging, you can edit the preview of data in a data flow by clicking "Debug Setting".

Examples of changing the data preview could be a row limit or file source in case you use source transformations.

When you select the staging linked service, you can use Azure Synapse Analytics as a source.

If you have parameters in your Data Flow or any of its referenced datasets, you can specify what values to use

during debugging by selecting the Parameters tab.

During debugging, sinks are not required and are ignored in the dataflow. If you want to test and write the transformed data to your sink, you can execute the data flow from a pipeline and use the debug execution from the pipeline.

As mentioned, within Azure Data Factory it is possible to only debug till a certain point or an activity.

In order to do so, you can use a breakpoint on the activity till where you want to test and then select Debug.

A Debug Until option appears as an empty red circle at the upper right corner of the element.

After you select the Debug Until option, it changes to a filled red circle to indicate the breakpoint is enabled.

Azure Data Factory will then make sure that the test only runs until that breakpoint activity in the pipeline. Especially when you want to test only a subset of the activities in a pipeline, this feature is useful.

In most scenarios the debug features in Azure Data Factory are sufficient.

However, sometimes it is necessary to test changes in a pipeline in a cloned sandbox environment.

A use-case to do so could be when you have parameterized ETL pipelines that you'd like to test how they would behave when they trigger a file arrival versus over tumbling time window.

In that case the cloning of a sandbox environment might be more suitable.

A good thing to know about Azure Data Factory might be that since it's mostly only charged by the number of runs, a second Data Factory doesn't have to lead to additional charges.

Monitoring debug runs

In order to monitor debug runs, you can check the output tab, but only for the most recent run that occurred in the browsing session, since it won't show the history.

If you would like to get a view of the history of debug runs, or see all the active debug runs, you can navigate to the monitor tab.

One thing to take in mind is that the Azure Data Factory service only keeps debug run history for 15 days. In relation to monitoring your data flow debug sessions, you would also navigate to the monitor tab.

Add parameters to data factory components

Parameterize linked services in Azure Data Factory

Within Azure Data Factory, it is possible to parameterize a linked service in which you can pass through dynamic values while at run time.

A use-case for this situation could be connecting to several different databases that are on the same SQL server, in which you might think about parameterizing the database name in the linked service definition. The benefit of doing so, is that you don't have to create a single linked service for each database that is on the same SQL Server.

It is also possible to parameterize other properties of the linked service like a Username.

If you decide to parameterize linked services in Azure Data Factory, you have the possibility to do so in the Azure Data Factory User Interface, the Azure portal or a programming interface to your liking.

If you choose to author the linked service through the User Interface, Data Factory can provide you with built-in parameterization for some of the connectors:

- Amazon Redshift
- Azure Cosmos DB (SQL API)
- Azure Database for MySQL
- Azure SQL Database
- Azure Synapse Analytics (formerly SQL DW)
- MySQL
- Oracle
- SQL Server
- Generic HTTP
- Generic REST

If you navigate to the creation/edit blade of the linked service, you will find the options for the parameterizing.

The screenshot shows the 'Edit linked service (Azure SQL Database)' blade in the Azure Data Factory UI. The 'Parameters' section is highlighted with a red box. It contains a table with one row:

NAME	TYPE	DEFAULT VALUE
Name	String	Value

If you cannot use the built-in parameterization since you're using a different type of connector, you are able to edit the JSON through the UI:

In linked service creation/edit blade -> expand "Advanced" at the bottom -> check "Specify dynamic contents in JSON format" checkbox -> specify the linked service JSON payload.

The screenshot shows the 'Edit linked service (Azure SQL Database)' blade in the Azure Data Factory UI. The 'Advanced' section is highlighted with a red box. It contains a checked checkbox for 'Specify dynamic contents in JSON format' and a JSON code editor showing the JSON payload:

```
{
  "type": "Microsoft.DataFactory/factories/linkedservices"
}
```

Or, after you create a linked service without parameterization, in Management hub -> Linked services -> find the specific linked service -> click "Code" (button "{}") to edit the JSON.

Global parameters in Azure Data Factory

Setting Global parameters in an Azure Data Factory pipeline, allows you to use these constants for consumption in pipeline expressions.

A use-case for setting global parameters is when you have multiple pipelines where the parameters names and values are identical.

If you use the continuous integration and deployment process with Azure Data Factory, the global parameters can be overridden if you wish so, for each and every environment that you have created.

Exercise creating global parameters in Azure Data Factory

To create a global parameter, go to the Global parameters tab in the Manage section. Select New to open the creation side-nav.

In the side-nav, enter a name, select a data type, and specify the value of your parameter.

After a global parameter is created, you can edit it by clicking the parameter's name. To alter multiple parameters at once, select Edit all.

Using global parameters in a pipeline

When using global parameters in a pipeline in Azure Data Factory, it is mostly referenced in pipeline expressions.

For example, if a pipeline references to a resource like a dataset or data flow, you can pass down the global parameter value through the resource parameter. The command or reference of global parameters in Azure Data Factory flows as follows: `pipeline().globalParameters.<parameterName>`.

Global parameters in CI/CD

When you integrate global parameters in a pipeline using CI/CD with Azure Data Factory, you have two ways in order to do so:

- Include global parameters in the Azure Resource Manager template
- Deploy global parameters via a PowerShell script

In most CI/CD practices, it is beneficial to include global parameters in the Azure Resource Manager template.

The reason why it's recommended is of the native integration with CI/CD where global parameters are added as an Azure Resource Manager Template parameter due to changes in several environments that are worked in.

In order to enable global parameters in an Azure Resource Manager template, you navigate to the management hub.

You do have to be aware that once you add global parameters to an Azure Resource Manager template, it adds an Azure Data Factory level setting, which can override other settings like git configs.

The use case for deploying global parameters through a PowerShell script, could be because you might have the above mentioned settings enabled in an elevated environment like UAT or PROD.

Parameterize mapping dataflows

Within Azure Data Factory, you are able to use mapping data flows and therefore, enabling you to use parameters.

If you set parameters inside a data flow definition, you can use the parameters in expressions.

The parameter values will be set by the calling pipeline through the Execute Data Flow activity.

There are three options for setting the values in the data flow activity expressions:

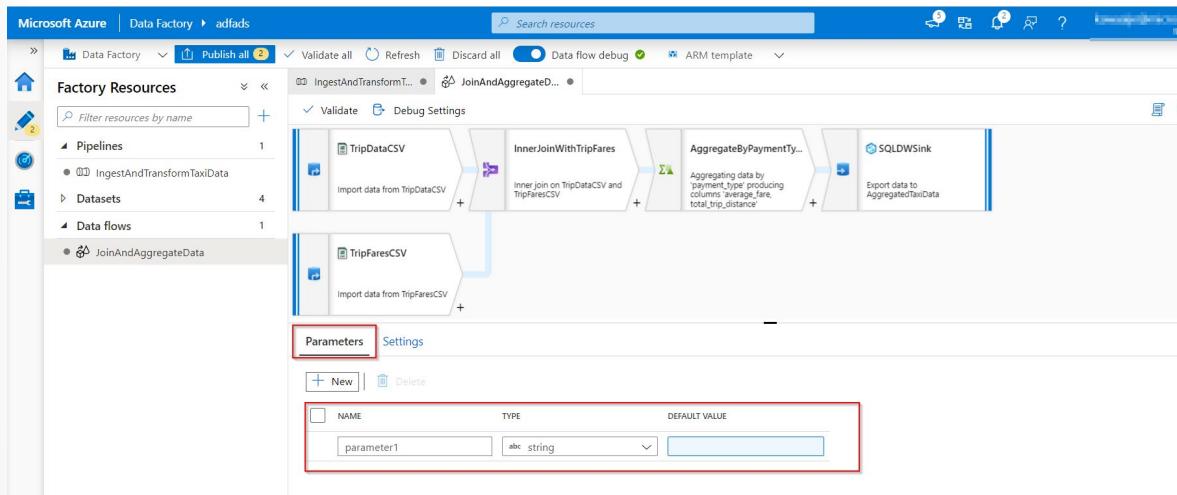
- Use the pipeline control flow expression language to set a dynamic value
- Use the data flow expression language to set a dynamic value
- Use either expression language to set a static literal value

The reason for parameterizing mapping data flows, is to make sure that your data flows are generalized, flexible, and reusable.

Create parameters in dataflow

To add parameters to your data flow, click on the blank portion of the data flow canvas to see the general properties. In the settings pane, you will see a tab called Parameter.

Select New to generate a new parameter. For each parameter, you must assign a name, select a type, and optionally set a default value.



Assign parameters from a pipeline in mapping dataflow

If you have created a data flow in which you have set parameters, it is possible to execute it from a pipeline using the Execute Data Flow Activity.

Once you have added the activity to the pipeline canvas, you'll find the data flow parameters in the activity's Parameters tab.

Assigning parameter values, ensures that you are able to use the parameters in a pipeline expression language or data flow expression language based on spark types. You can also combine the two, that is, pipeline and data flow expression parameters.

Exercise - Integrate a Notebook within Azure Synapse Pipelines

In this unit, you create an Azure Synapse Spark notebook to analyze and transform data loaded by a mapping data flow and store the data in a data lake. You create a parameter cell that accepts a string parameter that defines the folder name for the data the notebook writes to the data lake. You then add this notebook to a Synapse pipeline and pass the unique pipeline run ID to the notebook parameter so that you can later correlate the pipeline run with the data saved by the notebook activity. Finally, you use the Monitor hub in Synapse Studio to monitor the pipeline run, obtain the run ID, then locate the corresponding files stored in the data lake.

About Apache Spark and notebooks

Apache Spark is a parallel processing framework that supports in-memory processing to boost the performance of big-data analytic applications. Apache Spark in Azure Synapse Analytics is one of Microsoft's implementations of Apache Spark in the cloud.

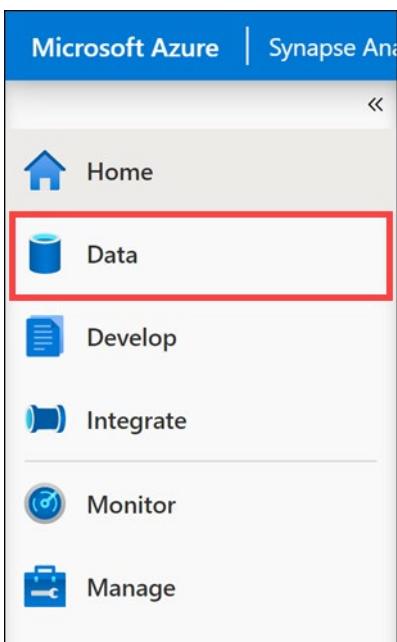
An Apache Spark notebook in Synapse Studio is a web interface for you to create files that contain live code, visualizations, and narrative text. Notebooks are a good place to validate ideas and use quick experiments to get insights from your data. Notebooks are also widely used in data preparation, data visualization, machine learning, and other Big Data scenarios.

Create a Synapse Spark notebook

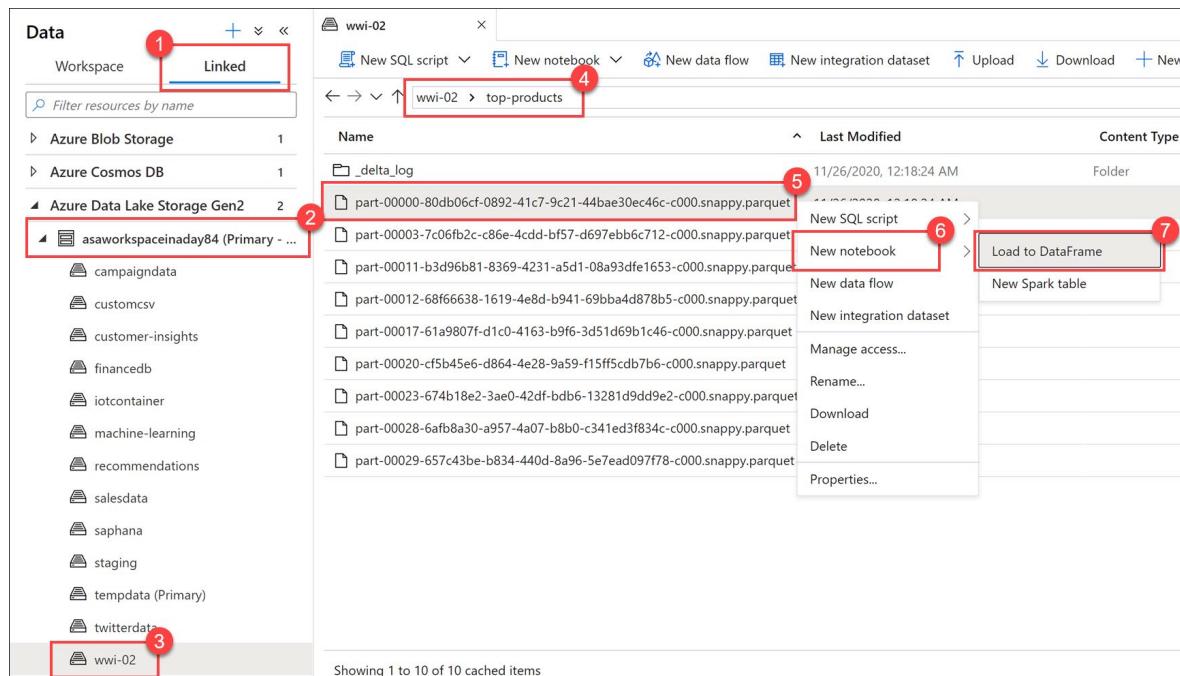
Suppose you created a Mapping Data flow in Synapse Analytics to process, join, and import user profile data. Now you want to find the top 5 products for each user, based on which ones are both preferred and top, and have the most purchases in the past 12 months. Then, you want to calculate the top 5 products overall.

In this step, you create a Synapse Spark notebook to make these calculations.

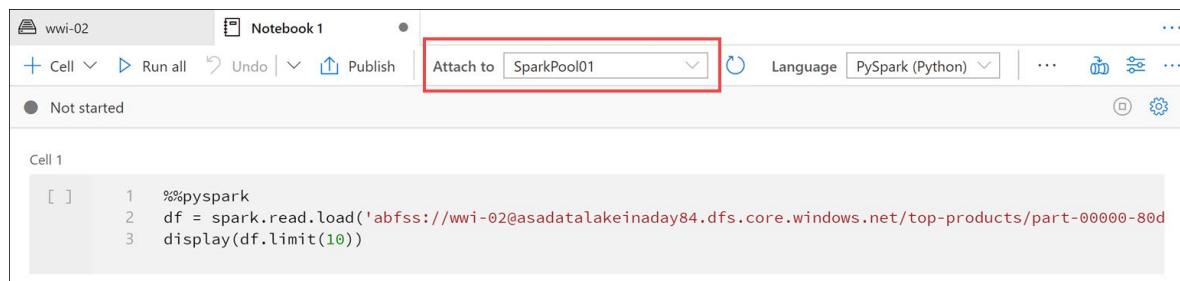
1. Open Synapse Analytics Studio (<https://web.azuresynapse.net/>), and then navigate to the **Data** hub.



2. Select the **Linked** tab (1) and expand the **primary data lake storage account** (2) underneath the **Azure Data Lake Storage Gen2**. Select the **wwi-02** container (3) and open the **top-products** folder (4). Right-click on any Parquet file (5), select the **New notebook** menu item (6), then select **Load to DataFrame** (7). If you don't see the folder, select Refresh above.



3. Make sure the notebook is attached to your Spark pool.



4. Replace the Parquet file name with `*.parquet`(1) to select all Parquet files in the `top-products` folder. For example, the path should be similar to: `abfss://wwi-02@YOUR_DATA LAKE_NAME.dfs.core.windows.net/top-products/*.parquet`.

The screenshot shows a Jupyter Notebook cell labeled 'Cell 1'. The code in the cell is:

```
Cell 1
1 %%pyspark
2 df = spark.read.load('abfss://wwi-02@asadatalakeinaday84.dfs.core.windows.net/top-products/*.parquet', format='parquet')
3 display(df.limit(10))
```

5. Select **Run all** on the notebook toolbar to execute the notebook.

Cell 1

```

1 %%pyspark
2 df = spark.read.load('abfss://wwi-02@asadatalakeinaday84.dfs.core.windows.net/top-products/*.parquet')
3 display(df.limit(10))

```

Command executed in 2mins 35s 588ms by joel on 11-26-2020 00:53:24.571 -05:00

> Job execution Succeeded Spark 2 executors 8 cores

[View in monitoring](#) [Open Spark UI](#)

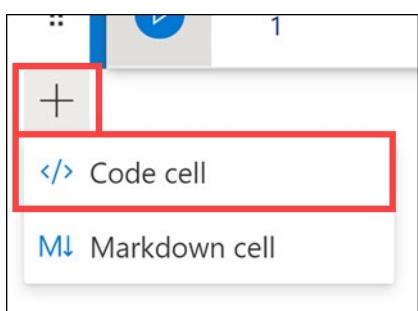
View [Table](#) [Chart](#)

visitorId	productId	itemsPurchased...	preferredProduct...	userId	isTopProduct	isPreferredProd...
2717		2717	148	false	true	
4002		4002	148	false	true	
1716		1716	148	false	true	
4520		4520	148	false	true	
951		951	148	false	true	
1817		1817	148	false	true	
2634		2634	463	false	true	
2795		2795	463	false	true	

NOTE: The first time you run a notebook in a Spark pool, Synapse creates a new session. This can take approximately 3-5 minutes.

NOTE: To run just the cell, either hover over the cell and select the *Run cell* icon to the left of the cell, or select the cell then type **Ctrl+Enter** on your keyboard.

6. Create a new cell underneath by selecting the + button and selecting the **Code cell** item. The + button is located beneath the notebook cell on the left. Alternatively, you can also expand the + Cell menu in the Notebook toolbar and select the **Code cell** item.



7. Enter and execute the following in the new cell to populate a new dataframe called `topPurchases`, create a new temporary view named `top_purchases`, and show the first 100 rows:

```

topPurchases = df.select(
    "UserId", "ProductId",
    "ItemsPurchasedLast12Months", "IsTopProduct",
    "IsPreferredProduct")

# Populate a temporary view so we can query from SQL
topPurchases.createOrReplaceTempView("top_purchases")

```

```
topPurchases.show(100)
```

The output should look similar to the following:

UserId	ProductId	ItemsPurchasedLast12Months	IsTopProduct	IsPreferredProduct
148	2717	null	false	true
148	4002	null	false	true
148	1716	null	false	true
148	4520	null	false	true
148	951	null	false	true
148	1817	null	false	true
463	2634	null	false	true
463	2795	null	false	true
471	1946	null	false	true
471	4431	null	false	true
471	566	null	false	true
471	2179	null	false	true
471	3758	null	false	true
471	2434	null	false	true
471	1793	null	false	true
471	1620	null	false	true
471	1572	null	false	true
833	957	null	false	true
833	3140	null	false	true
833	1087	null	false	true

8. Execute the following in a new cell to create a new temporary view by using SQL:

```
%%sql
```

```
CREATE OR REPLACE TEMPORARY VIEW top_5_products
AS
select UserId, ProductId, ItemsPurchasedLast12Months
from (select *,
           row_number() over (partition by UserId order by ItemsPurchasedLast12Months desc) as seqnum
      from top_purchases
     ) a
where seqnum <= 5 and IsTopProduct == true and IsPreferredProduct = true
order by a.UserId
```

Note that there is no output for the above query. The query uses the `top_purchases` temporary view as a source and applies a `row_number()` over method to apply a row number for the records for each user where `ItemsPurchasedLast12Months` is greatest. The `where` clause filters the results so we only retrieve up to five products where both `IsTopProduct` and `IsPreferredProduct` are set to true. This gives us the top five most purchased products for each user where those products are also identified as their favorite products, according to their user profile stored in Azure Cosmos DB.

9. Execute the following in a new cell to create and display a new DataFrame that stores the results of the top 5 products temporary view you created in the previous cell:

```
top5Products = sqlContext.table("top_5_products")
```

```
top5Products.show(100)
```

You should see an output similar to the following, which displays the top five preferred products per user:

Cell 5

```
[26] 1 top5Products = sqlContext.table("top_5_products")
2
3 top5Products.show(100)
```

User Id	Product Id	Items Purchased Last 12 Months
80000	2069	93
80000	2069	93
80000	2069	93
80000	2069	93
80000	2069	93
80000	2069	93
80001	1812	93
80001	1812	93
80001	1812	93
80001	1812	93
80001	1812	93
80002	1256	90
80002	1256	90
80002	4987	88
80002	3190	92
80002	3190	92
80003	295	91
80003	638	97

10. Calculate the top five products overall, based on those that are both preferred by customers and purchased the most. To do this, execute the following in a new cell:

```
top5ProductsOverall = (top5Products.select("ProductId", "ItemsPurchasedLast12Months")
    .groupBy("ProductId")
    .agg( sum("ItemsPurchasedLast12Months").alias("Total") )
    .orderBy( col("Total").desc() )
    .limit(5))
```

```
top5ProductsOverall.show()
```

In this cell, we grouped the top five preferred products by product ID, summed up the total items purchased in the last 12 months, sorted that value in descending order, and returned the top five results. Your output should be similar to the following:

+	-----	+	-----	+
	ProductId		Total	
+	-----	+	-----	+

```
| 2107| 4538|
| 4833| 4533|
| 347| 4523|
| 3459| 4233|
| 4246| 4155|
+-----+-----+
```

Create a parameter cell

Azure Synapse pipelines look for the parameters cell and treat this cell as defaults for the parameters passed in at execution time. The execution engine will add a new cell beneath the parameters cell with input parameters in order to overwrite the default values. When a parameters cell isn't designated, the injected cell will be inserted at the top of the notebook.

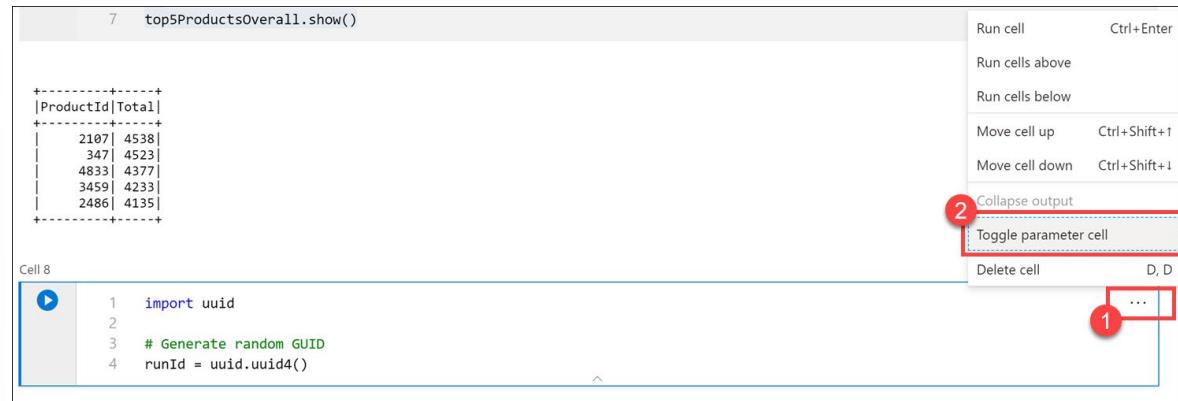
1. We are going to execute this notebook from a pipeline. We want to pass in a parameter that sets a `runId` variable value that will be used to name the Parquet file. Execute the following in a new cell:

```
import uuid

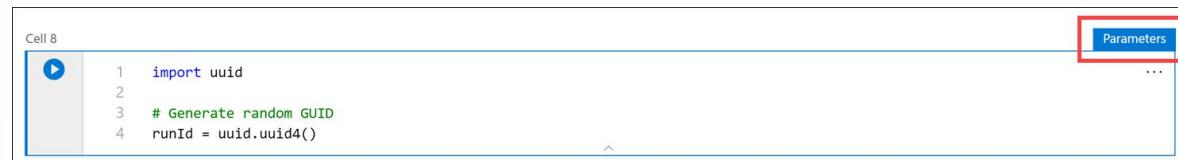
# Generate random GUID
runId = uuid.uuid4()
```

We are using the `uuid` library that comes with Spark to generate a random GUID. We want to override the `runId` variable with a parameter passed in by the pipeline. To do this, we need to toggle this as a parameter cell.

2. Select the actions ellipses (...) on the top-right corner of the cell (1), then select **Toggle parameter cell** (2).



After toggling this option, you will see the **Parameters** tag on the cell.



3. Paste the following code in a new cell to use the `runId` variable as the Parquet filename in the `/top5-products/` path in the primary data lake account. Replace `YOUR_DATALAKE_NAME` in the

path with the name of your primary data lake account. To find this, scroll up to **Cell 1** at the top of the page **(1)**. Copy the data lake storage account from the path **(2)**. Paste this value as a replacement for **YOUR_DATALAKE_NAME** in the path **(3)** inside the new cell, then execute the cell.

```
%%pyspark
```

```
top5ProductsOverall.write.parquet('abfss://wwi-02@YOUR_DATALAKE_NAME.dfs.core.windows.net/
top5-products/' + str(runId) + '.parquet')
```

The screenshot shows a Jupyter Notebook interface with three cells:

- Cell 1:** Contains the following code:


```
[22] 1 %%pyspark
2 data_path = spark.read.load('abfss://wwi-02@asadatalakeinaday84.dfs.core.windows.net/top-products/*.parquet', format='parquet')
3 display(data_path.limit(10))
```
- Cell 2:** Contains the following code:


```
[29] 1 import uuid
2
3 # Generate random GUID
4 runId = uuid.uuid4()
```
- Cell 9:** Contains the modified code from Cell 1, where the path is replaced by the value from Cell 2:


```
Cell 9 [29] 1 top5ProductsOverall.write.parquet('abfss://wwi-02@asadatalakeinaday84.dfs.core.windows.net/top5-products/' + str(runId) ...)
```

4. Verify that the file was written to the data lake. Navigate to the **Data** hub and select the **Linked** tab **(1)**. Expand the primary data lake storage account and select the **wwi-02** container **(2)**. Navigate to the **top5-products** folder **(3)**. You should see a folder for the Parquet file in the directory with a GUID as the file name **(4)**.

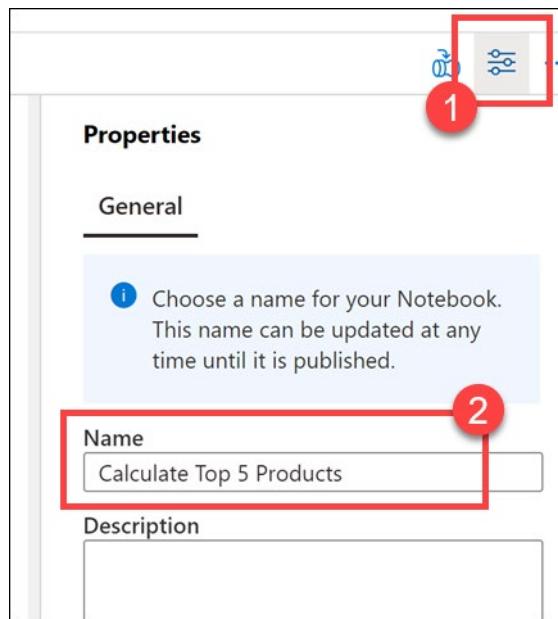
The screenshot shows the Data hub in Azure Data Studio. On the left, the 'Linked' tab is selected (1). On the right, the 'wwi-02' container is expanded (2), showing its contents. The 'top5-products' folder is selected (3), and it contains a single file named '12fe8c61-0998-4124-bf3c-fa751d8ce088.parquet' (4).

The Parquet write method on the dataframe in the Notebook cell created this directory since it did not previously exist.

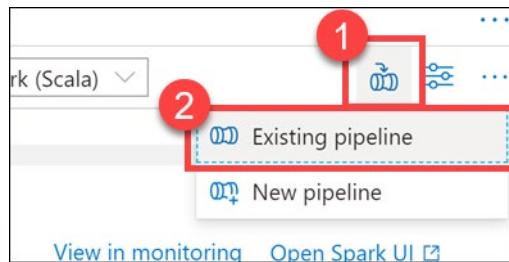
Add the Notebook to a Synapse pipeline

Referring back to the Mapping Data Flow we mentioned at the beginning of the exercise, suppose you want to execute this notebook after the Data Flow runs as part of your orchestration process. To do this, you add this notebook to a pipeline as a new Notebook activity.

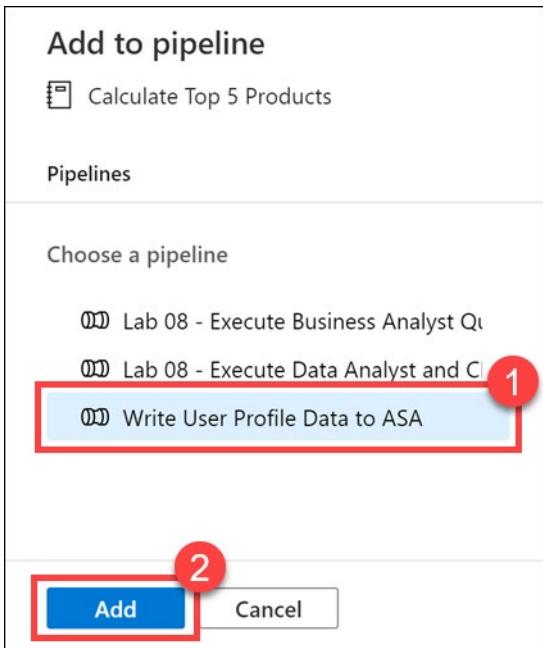
1. Return to the notebook. Select the **Properties** button (1) at the top-right corner of the notebook, then enter Calculate Top 5 Products for the **Name** (2).



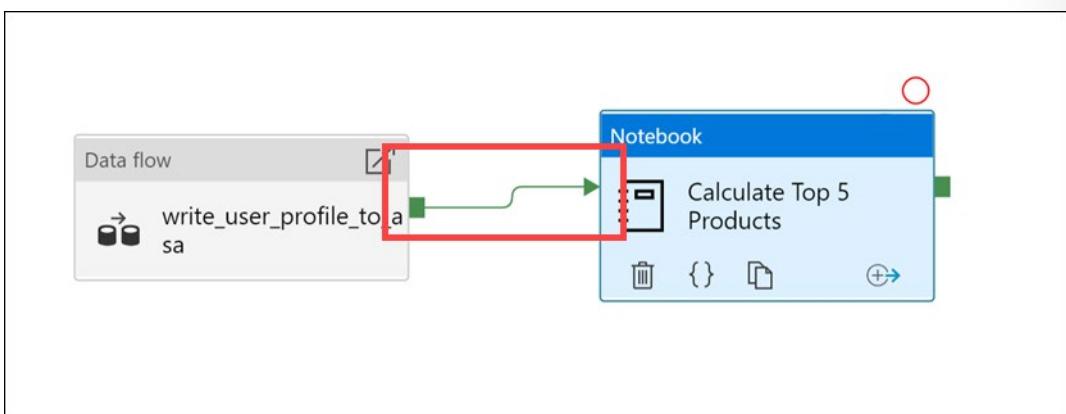
2. Select the **Add to pipeline** button (1) at the top-right corner of the notebook, then select **Existing pipeline** (2).



3. Select the **Write User Profile Data to ASA** pipeline (1), then select **Add *2**.

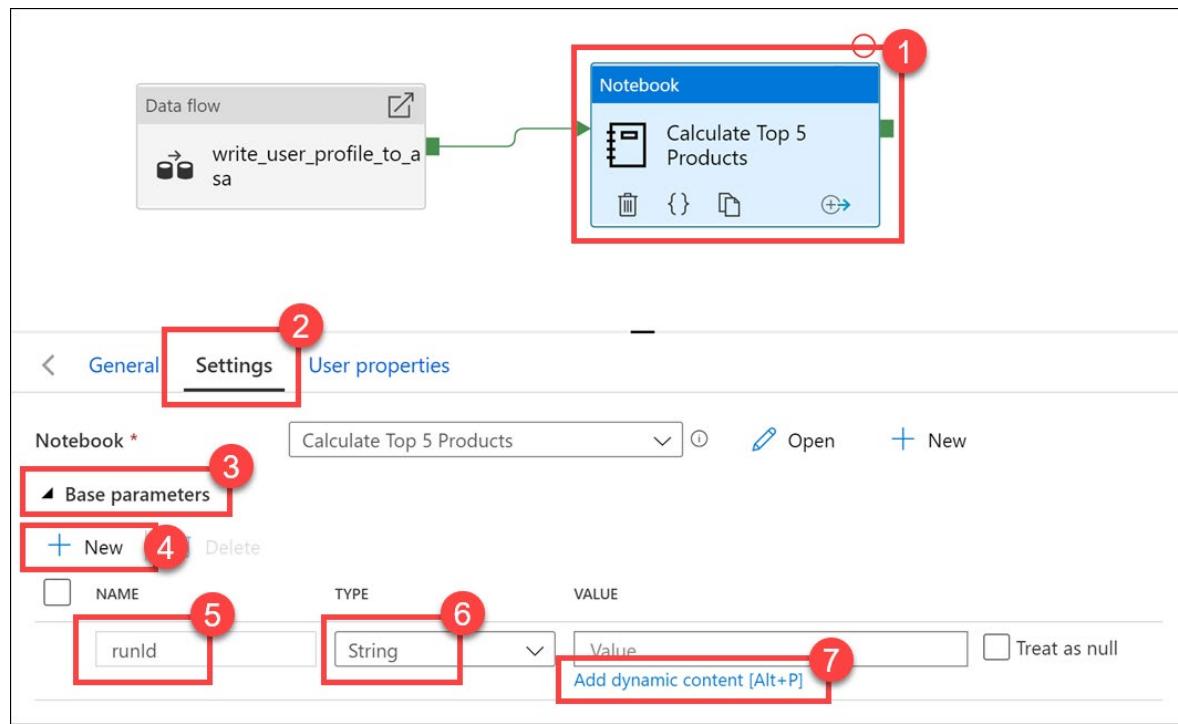


4. Synapse Studio adds the Notebook activity to the pipeline. Rearrange the **Notebook activity** so it sits to the right of the **Data flow activity**. Select the **Data flow activity** and drag a **Success** activity pipeline connection **green box** to the **Notebook activity**.

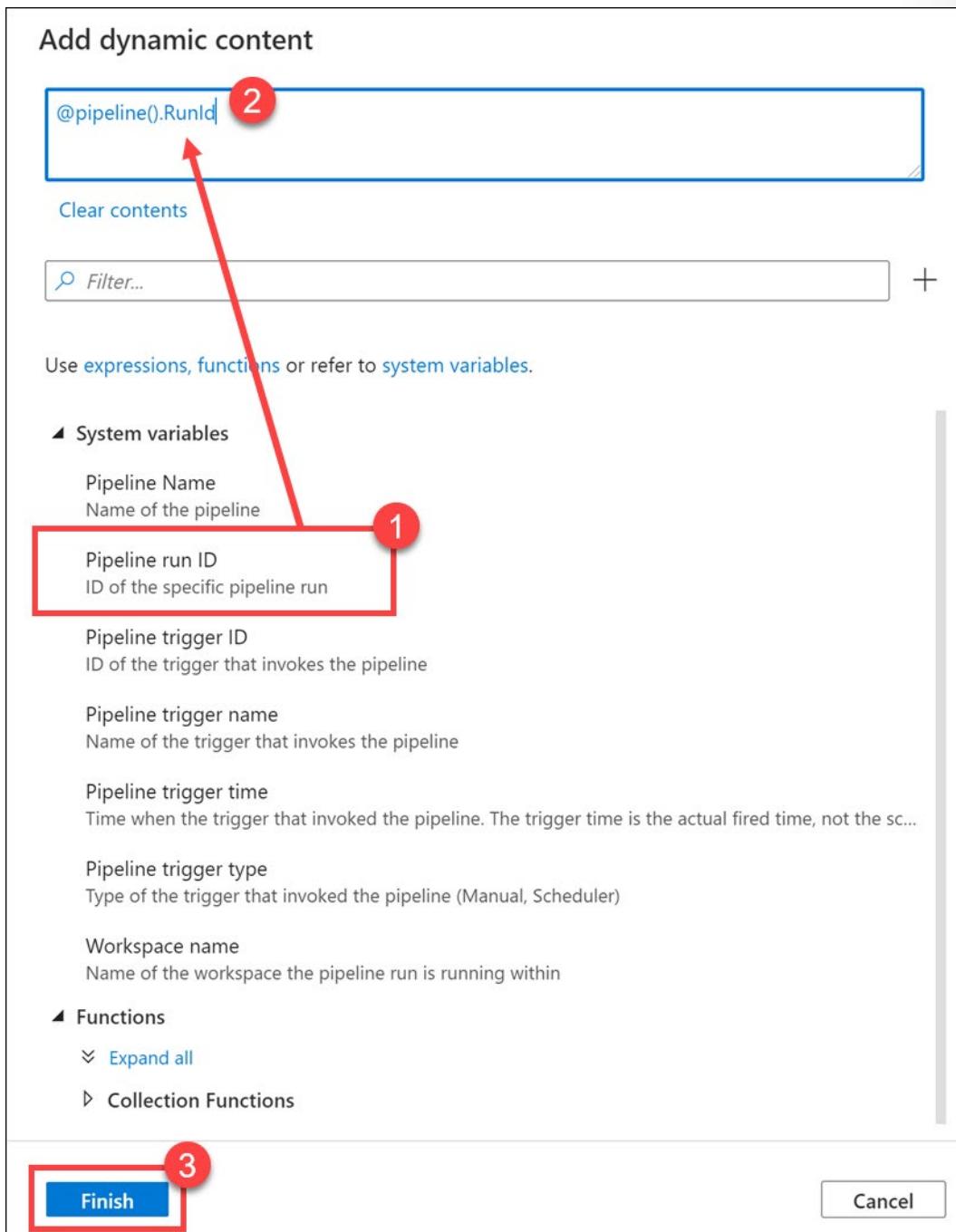


The Success activity arrow instructs the pipeline to execute the Notebook activity after the Data flow activity successfully runs.

5. Select the **Notebook activity** (1), select the **Settings** tab (2), expand **Base parameters** (3), and select **+ New** (4). Enter **runId** in the **Name** field (5). Select **String** for the **Type** (6). For the **Value**, select **Add dynamic content** (7).



6. Select **Pipeline run ID** under **System variables** (1). This adds `@pipeline().RunId` to the dynamic content box (2). Select **Finish** (3) to close the dialog.

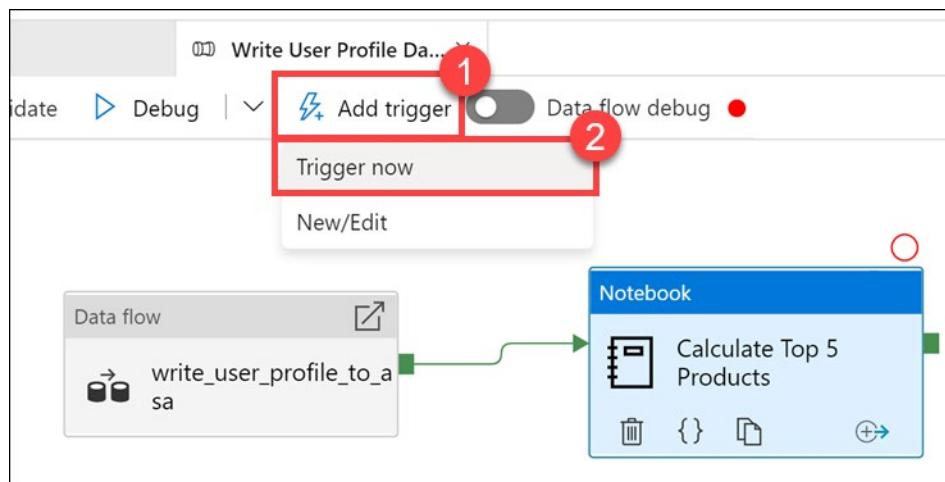


The Pipeline run ID value is a unique GUID assigned to each pipeline run. We will use this value for the name of the Parquet file by passing this value in as the `runId` Notebook parameter. We can then look through the pipeline run history and find the specific Parquet file created for each pipeline run.

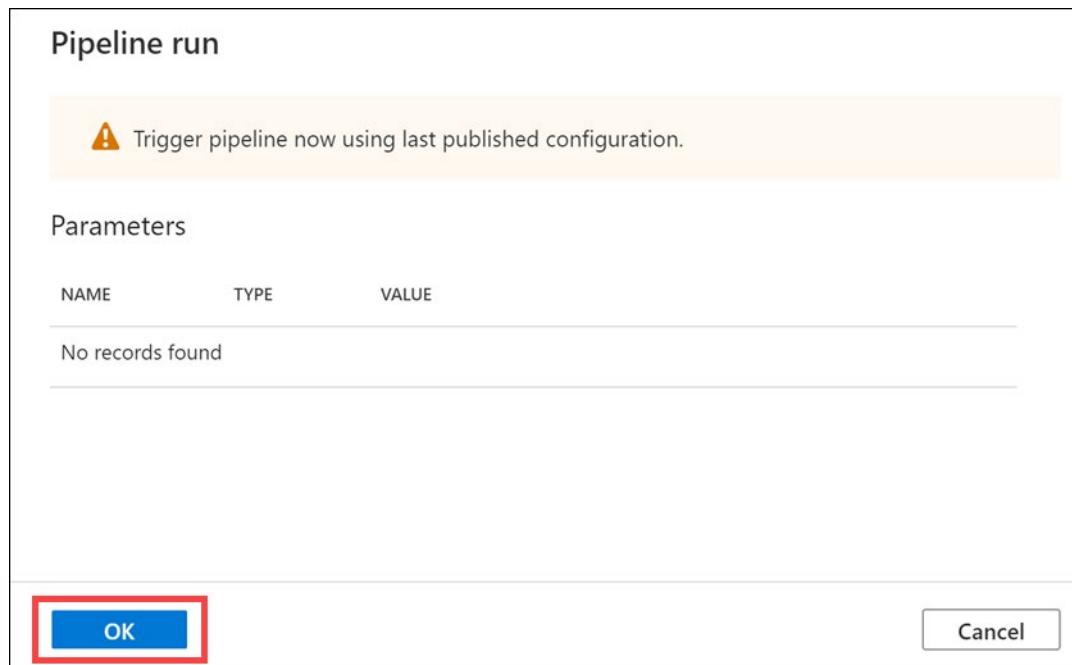
7. Select **Publish all** then **Publish** to save your changes.



8. After publishing is complete, select **Add trigger (1)**, then **Trigger now (2)** to run the updated pipeline.



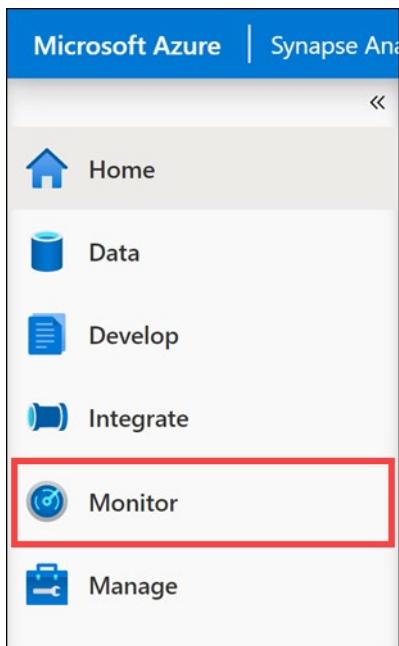
9. Select **OK** to run the trigger.



Monitor the pipeline run

The Monitor hub lets you monitor current and historical activities for SQL, Apache Spark, and Pipelines.

1. Navigate to the **Monitor** hub.



2. Select **Pipeline runs (1)** and wait for the pipeline run to successfully complete (2). You may need to refresh (3) the view.

The screenshot shows the 'Pipeline runs' page. The left sidebar has 'Integration' selected, with 'Pipeline runs' highlighted by a red box. The main area shows a table of pipeline runs. Two rows are visible, both with a status of 'Succeeded' (indicated by a green circle with a checkmark and a red box). The table includes columns for Pipeline name, Run start, Run end, Duration, Triggered by, Status, and Run. The top right of the page has a 'Refresh' button, which is also highlighted with a red box.

Run	Pipeline name	Run start	Run end	Duration	Triggered by	Status
1	Write User Profile Data to ASA	11/26/20, 3:33:48 AM	11/26/20, 3:45:43 AM	00:11:55	Manual trigger	Succeeded
2	Write User Profile Data to ASA	11/26/20, 3:16:33 AM	11/26/20, 3:21:26 AM	00:04:53	Manual trigger	Succeeded

3. Select the name of the pipeline to view the pipeline's activity runs.

The screenshot shows the 'Pipeline runs' page with a single row of data. The pipeline name is 'Write User Profile Data to ASA', which is highlighted with a red box. The run started at 11/26/20, 3:33:48 AM. The top navigation bar shows 'Triggered' selected, along with 'Rerun', 'Cancel', and 'Refresh' buttons.

4. Notice both the **Data flow** activity, and the new **Notebook** activity (1). Make note of the **Pipeline run ID** value (2). We will compare this to the Parquet file name generated by the notebook. Select the **Calculate Top 5 Products** Notebook name to view its details (3).

The screenshot shows the 'Write User Profile Data to ASA' pipeline. At the top, there are two tabs: 'List' (selected) and 'Gantt'. Below the tabs are buttons for 'Rerun', 'Rerun from activity', 'Rerun from failed activity', and 'Refresh'. The main area displays a sequence of activities: a 'Data flow' activity named 'write_user_profile_to_asa' followed by a 'Notebook' activity named 'Calculate Top 5 Products'. A red box highlights the 'Notebook' activity, and a red circle labeled '1' is placed above it. Below the activities is a section titled 'Activity runs'. It shows a single run entry: 'Pipeline run ID 16bb8447-824a-4147-b7ea-348505ebdc44'. This entry is also highlighted with a red box and labeled '2'. At the bottom of the 'Activity runs' section is a table with the following data:

Activity name	Activity type	Run start ↑	Duration	Status	Integration runtime
Calculate Top 5 Products	SynapseNotebook	11/26/20, 3:39:57 AM	00:05:45	Succeeded	DefaultIntegrationRuntime (East US 2)
write_user_profile_to_asa	ExecuteDataFlow	11/26/20, 3:33:50 AM	00:06:07	Succeeded	DefaultIntegrationRuntime (East US 2)

A red box highlights the 'Calculate Top 5 Products' row, and a red circle labeled '3' is placed next to the 'Activity name' column header.

5. Here we see the Notebook run details. You can select the **Playback** button (1) to watch a playback of the progress through the **jobs** (2). At the bottom, you can view the **Diagnostics** and **Logs** with different filter options (3). To the right, we can view the run details, such as the duration, Livy ID, Spark pool details, etc. Select the **View details** link on a **job** to view its details (5).

The screenshot shows the Spark Application UI for application ID application_1606380084300_0001. The main interface displays the DAG visualization with stages 0 through 5. Stage 4 is expanded, showing tasks 200 and 201. A red box labeled 1 highlights the 'Playback' button at the top center. A red box labeled 2 highlights Stage 1. A red box labeled 3 highlights the 'Diagnostics' section. A red box labeled 4 highlights the 'Details' section. A red box labeled 5 highlights Stage 4.

Application

Application ID: application_1606380084300_0001

Queued duration: 0s

Running duration: 5m 45s

Livy ID: 4

Submitter: ee20d9e7-6295-4240-ba3f-c3784616c565

Executors: 2

Spark pool:

- Name: SparkPool01

DAG Visualization

Attempts 1 of 1

Stage 0 (Job 0): Tasks 1, Duration: 6s 510ms, Rows: 0, Data read: 0Byte, Data written: 0Byte. View details

Stage 1 (Job 1): Tasks 1, Duration: 6s 892ms, Rows: 4096, Data read: 1.6MBs, Data written: 0Byte. View details

Stage 2 (Job 2): Tasks 8, Duration: 2s 469ms, Rows: 1622203, Data read: 5.7MBs, Data written: 0Byte. View details

Stage 3 (Job 3): Tasks 8, Duration: 2s 539ms, Rows: 3244406, Data read: 5.7MBs, Data written: 10.1MBs. View details

Stage 4 (Job 4): Tasks 200, Duration: 3s 90ms, Rows: 1622201, Data read: 10.1MBs, Data written: 0Byte. View details

Stage 5 (Job 5): Tasks 8, Duration: 6s 680ms, Rows: 1622203, Data read: 875.3KBs, Data written: 0Byte. View details

Diagnostics

- > Failed jobs
- > Data skew
- > Time skew
- > Executor utilization

Logs

- The Spark application UI opens in a new tab where we can see the stage details. Expand the **DAG Visualization** to view the stage details.

Details for Stage 4 (Attempt 0)

Total Time Across All Tasks: 18 s
Locality Level Summary: Node local: 200
Shuffle Read: 10.1 MB / 1622203

▼ DAG Visualization

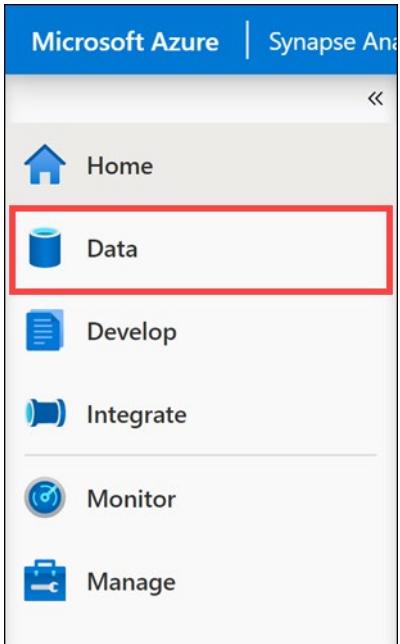
```
graph TD; A[ShuffledRowRDD [13] showString at NativeMethodAccessorImpl.java:0] --> B[MapPartitionsRDD [14] showString at NativeMethodAccessorImpl.java:0]; B --> C[MapPartitionsRDD [15] showString at NativeMethodAccessorImpl.java:0]; C --> D[MapPartitionsRDD [16] showString at NativeMethodAccessorImpl.java:0]; D --> E[MapPartitionsRDD [17] showString at NativeMethodAccessorImpl.java:0]; E --> F[MapPartitionsRDD [18] showString at NativeMethodAccessorImpl.java:0]
```

▶ Show Additional Metrics
▶ Event Timeline

Summary Metrics for 200 Completed Tasks

Metric	Min	25th percentile	Median
Duration	20 ms	42 ms	57 ms
GC Time	0 ms	0 ms	0 ms
Shuffle Read Size / Records	43.9 KB / 6471	50.2 KB / 7648	51.5 KB / 8057

7. Navigate back to the **Data** hub.

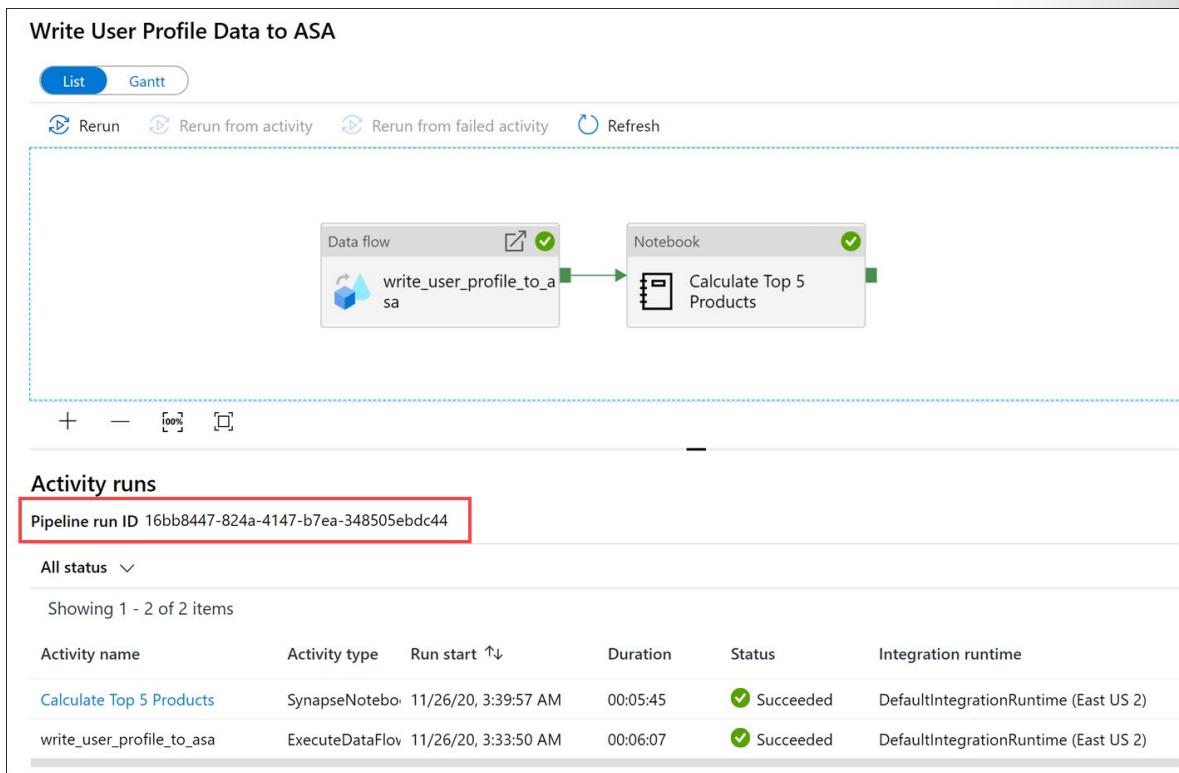


8. Select the **Linked** tab (1), select the **wwi-02** container (2) on the primary data lake storage account, navigate to the **top5-products** folder (3), and verify that a folder exists for the Parquet file whose name matches the **Pipeline run ID**.

The screenshot shows the Azure Data Explorer interface. On the left, a sidebar titled "Data" lists various Azure services and a workspace named "asaworkspaceinaday84 (Primary - ...)" which is expanded to show sub-folders like campaigndata, customcsv, customer-insights, etc. A red box labeled "1" highlights the "Linked" button at the top of the sidebar. A red box labeled "2" highlights the "wwi-02" folder in the workspace list.

On the right, a main pane displays a pipeline run history. At the top, there are buttons for "New SQL script", "New data flow", and "New integration". Below them, a breadcrumb navigation bar shows the path: "wwi-02 > top5-products". A red box labeled "3" highlights this breadcrumb. The main area is titled "Name" and lists two items: "16bb8447-824a-4147-b7ea-348505ebdc44.parquet" and "cbb0540e-afe2-4c84-9ced-7f45d3424205.parquet". A red box labeled "4" highlights the first item in the list.

As you can see, we have a file whose name matches the **Pipeline run ID** we noted earlier:



These values match because we passed in the Pipeline run ID to the `runId` parameter on the Notebook activity.

Execute data factory packages

Azure Data Factory enables you to execute packages for SSIS.

With Azure Data Factory, you are able to provision an Azure-SQL Server Integration Services (SSIS) integration runtime (IR).

An Azure-SSIS IR supports:

- Running packages that are or to be deployed into SSIS catalog (SSISDB) hosted by your Azure SQL Database server or Managed Instance in Project Deployment Model
- Running packages that are or to be deployed into file system, Azure Files, or SQL Server database (MSDB) hosted by your Azure SQL Managed Instance in Package Deployment Model

Once the Azure-SSIS IR is provisioned, the same familiar tools for deployment and running the packages in Azure can be used.

Most of the familiar tools such as SQL Server Data Tools (SSDT), SQL Server Management Studio (SSMS), Azure Data Studio, and command-line utilities are Azure- enabled, and therefore ready to be used.

Using SSDT allows you to check and assess the Azure Cloud compatibility with Azure-SSIS Integration Runtime in Azure Data Factory of the SSIS packages you might already be running locally. This feature comes in handy when you want to test existing packages before an actual lift and shift or migration can take place to Azure. If you want to develop new packages to run in Azure, it's also good to test them with this feature.

Knowledge check

Question 1

What is a supported connector for built-in parameterization?

- Azure Data Lake Storage Gen2.
- Azure Synapse Analytics.
- Azure Key Vault.

Question 2

What is an example of a branching activity used in control flows?

- Thelf-condition.
- Until-condition.
- Lookup- condition.

Summary

Now you have learned how to orchestrate data movement and transformations in Azure Data Factory.

Since you have completed this module you are now able to:

- Understand the data factory control flow
- Work with data factory pipelines
- Debug data factory pipelines
- Add parameters to data factory components
- Execute data factory packages

Module Lab information

Lab 7 - Integrate data from Notebooks with Azure Data Factory or Azure Synapse Pipelines

- **Estimated Time:** 50 - 60 minutes
- **Lab files:** The files for this lab are located in the *Instructions\Labs\09* folder.

Lab overview

In the lab, the students will create a notebook to query user activity and purchases that they have made in the past 12 months. They will then add the notebook to a pipeline using the new Notebook activity and execute this notebook after the Mapping Data Flow as part of their orchestration process. While configuring this the students will implement parameters to add dynamic content in the control flow and validate how the parameters can be used.

Lab objectives

After completing this lab, you will be able to:

- Integrate data from Notebooks with Azure Data Factory or Azure Synapse Pipelines

Module summary

module-summary

In this module, students have learned how to orchestrate data movement and transformations in Azure Data Factory or Azure Synapse Pipeline. Students understand how Azure Data Factory or Azure Synapse pipelines can orchestrate large scale data movement by using other Azure Data Platform and Machine Learning technologies.

Learning objectives

In this module, you have learned to:

- Understand how to add an activity to the control flow to orchestrate data from other technologies
- Understand how to use parameters in Azure Data Factory/Synapse pipelines

Post Course Review

After the course, consider watching the video on channel 9 that provides a summary of performing **Iterative development and debugging with Azure Data Factory**¹. Note that the content in this video also applies to Azure Synapse Pipelines too.

Important

Remember

Should you be working in your own subscription while running through this content, it is best practice at the end of a project to identify whether or not you still need the resources you created. Resources left running can cost you money.

You can delete resources one by one, or just delete the resource group to get rid of the entire set.

¹ <https://channel9.msdn.com/Shows/Azure-Friday/Iterative-development-and-debugging-with-Azure-Data-Factory>

Answers

Question 1

What is a supported connector for built-in parameterization?

- Azure Data Lake Storage Gen2.
- Azure Synapse Analytics.
- Azure Key Vault.

Explanation

Correct. Azure Synapse Analytics is a supported connector for built-in parameterization for Linked Services in Azure Data Factory.

Question 2

What is an example of a branching activity used in control flows?

- Thelf-condition.
- Until-condition.
- Lookup- condition.

Explanation

Correct. An example of a branching activity is Thelf-condition activity which is similar to an if-statement provided in programming languages.

Module 8 End-to-end security with Azure Synapse Analytics

Module introduction

module-introduction

In this module, you will learn how to approach and implement security to protect your data with Azure Synapse Analytics.

Learning objectives

In this module, you will:

- Secure a data warehouse
- Configure and manage secrets in Azure Key Vault
- Implement compliance controls for sensitive data

Secure a data warehouse in Azure Synapse Analytics

Introduction

In this module, you will learn how to approach and implement security to protect your data with Azure Synapse Analytics.

In this module, you will:

- Understand network security options for Azure Synapse Analytics
- Configure conditional access
- Configure Authentication
- Manage authorization through column and row level security
- Manage sensitive data with Dynamic Data masking
- Implement encryption in Azure Synapse Analytics
- Understand advanced data security options for Azure Synapse Analytics

Prerequisites

Before taking this module, it is recommended that the student is able to:

- Log into the Azure portal
- Create a Synapse Analytics Workspace
- Create an Azure Synapse Analytics SQL Pool

Understand network security options for Azure Synapse Analytics

There are a range of network security steps that you should consider to secure Azure Synapse Analytics. One of the first aspects that you will consider is securing access to the service itself. This can be achieved by creating the following network objects including:

- Firewall rules
- Virtual networks
- Private endpoints

Firewall rules

Firewall rules enable you to define the type of traffic that is allowed or denied access to an Azure Synapse workspace using the originating IP address of the client that is trying to access the Azure Synapse Workspace. IP firewall rules configured at the workspace level apply to all public endpoints of the workspace including dedicated SQL pools, serverless SQL pool, and the development endpoint.

You can choose to allow connections from all IP addresses as you are creating the Azure Synapse Workspaces, although this is not recommended as it does not allow for control access to the workspace.

Instead, within the Azure portal, you can configure specific IP address ranges and associate them with a rule name so that you have greater control.

The screenshot shows the Azure portal interface for managing firewalls in a Synapse workspace named 'asaworkspacetco'. The left sidebar contains navigation links for Overview, Activity log, Access control (IAM), Tags, Settings (with sub-options like SQL Active Directory admin, Properties, Locks), Analytics pools (SQL pools, Apache Spark pools), Security (Managed identities, Private endpoint connections ..., Azure SQL Auditing, Azure Defender for SQL), Monitoring (Alerts, Metrics), and Automation (Tasks (preview)). The 'Firewalls' option is highlighted with a red box. The main pane displays a summary message: 'The IPs listed below will have full access to Synapse workspace 'asaworkspacetco''. It includes an 'Allow Azure services and resources to access this workspace' toggle switch set to 'OFF', a 'Client IP address' field containing '151.224.130.178', and a table for defining firewall rules with columns for 'Rule name', 'Start IP', and 'End IP'. A 'Save' and 'Discard' button are at the top right.

Make sure that the firewall on your network and local computer allows outgoing communication on TCP ports 80, 443 and 1443 for Synapse Studio.

Also, you need to allow outgoing communication on UDP port 53 for Synapse Studio. To connect using tools such as SSMS and Power BI, you must allow outgoing communication on TCP port 1433.

Virtual networks

Azure Virtual Network (VNet) enables private networks in Azure. VNet enables many types of Azure resources, such as Azure Synapse Analytics, to securely communicate with other virtual networks, the internet, and on-premises networks. When you create your Azure Synapse workspace, you can choose to associate it to a Microsoft Azure Virtual Network. The Virtual Network associated with your workspace is managed by Azure Synapse. This Virtual Network is called a Managed workspace Virtual Network.

Using a managed workspace virtual network provides the following benefits:

- With a Managed workspace Virtual Network, you can offload the burden of managing the Virtual Network to Azure Synapse.

- You don't have to configure inbound NSG rules on your own Virtual Networks to allow Azure Synapse management traffic to enter your Virtual Network. Misconfiguration of these NSG rules causes service disruption for customers.
- You don't need to create a subnet for your Spark clusters based on peak load.
- Managed workspace Virtual Network along with Managed private endpoints protects against data exfiltration. You can only create Managed private endpoints in a workspace that has a Managed workspace Virtual Network associated with it.
- it ensures that your workspace is network isolated from other workspaces.

If your workspace has a Managed workspace Virtual Network, Data integration and Spark resources are deployed in it. A Managed workspace Virtual Network also provides user-level isolation for Spark activities because each Spark cluster is in its own subnet.

Dedicated SQL pool and serverless SQL pool are multi-tenant capabilities and therefore reside outside of the Managed workspace Virtual Network. Intra-workspace communication to dedicated SQL pool and serverless SQL pool use Azure private links. These private links are automatically created for you when you create a workspace with a Managed workspace Virtual Network associated to it.

You can only choose to enable managed virtual networks as you are creating the Azure Synapse Workspaces.

Home > New > Azure Synapse Analytics (workspaces preview) >

Create Synapse workspace

The screenshot shows the 'Networking' tab selected in the top navigation bar, which includes tabs for Basics, Security, Tags, and Summary. Below the tabs, a note says 'Configure networking settings for your workspace.' Under 'Allow connections from all IP addresses', there is a warning message: '⚠️ Azure Synapse Studio and other client tools will only be able to connect to the workspace endpoints if this setting is allowed. Connections from specific IP addresses or all Azure services can be allowed/disallowed after the workspace is provisioned.' A checkbox labeled 'Allow connections from all IP addresses' is checked. In the 'Managed virtual network' section, a note says 'Choose whether you want a Synapse-managed virtual network dedicated for your Azure Synapse workspace.' A checkbox labeled 'Enable managed virtual network' is unchecked and highlighted with a red border.

Private endpoints

Azure Synapse Analytics enables you to connect upto its various components through endpoints. You can set up managed private endpoints to access these components in a secure manner known as private links. This can only be achieved in an Azure Synapse workspace with a Managed workspace Virtual

Network. Private link enables you to access Azure services (such as Azure Storage and Azure Cosmos DB) and Azure hosted customer/partner services from your Azure Virtual Network securely.

When you use a private link, traffic between your Virtual Network and workspace traverses entirely over the Microsoft backbone network. Private Link protects against data exfiltration risks. You establish a private link to a resource by creating a private endpoint.

Private endpoint uses a private IP address from your Virtual Network to effectively bring the service into your Virtual Network. Private endpoints are mapped to a specific resource in Azure and not the entire service. Customers can limit connectivity to a specific resource approved by their organization. You can manage the private endpoints in the Azure Synapse Studio manage hub.

Name	Provisioning...	Approval st...	VNet name	Possible Linked Services	Linked resources
synapse-ws-sql-test	Succeeded	Approved	default	1	/subscriptions/...
synapse-ws-sqlOnDemand...	Succeeded	Approved	default	0	/subscriptions/...

Configure conditional access

Conditional access is a feature that enables you to define the conditions under which a user can connect to your Azure subscription and access services. Conditional access provides an additional layer of security that can be used in combination with authentication to strengthen the security access to your network.



<https://channel9.msdn.com/Shows/Docs-Azure/Azure-AD-Conditional-Access/player?format=ny>

Conditional Access policies at their simplest are if-then statements, if a user wants to access a resource, then they must complete an action. As an example, if a Data Engineer wishes to access services in Azure Synapse Analytics, they may be requested by the conditional access policy to perform an additional step of multi-factor authentication (MFA) to complete the authentication to get onto the service.

Conditional access policies use signals as a basis to determine if conditional access should first be applied. Common signals include:

- User or group membership names
- IP address information
- Device platforms or type

- Application access requests
- Real-time and calculated risk detection
- Microsoft Cloud App Security (MCAS)

Based on these signals, you can then choose to block access. The alternative is you can grant access, and at the same time request that the user perform an additional action including:

- Perform Multi-Factor authentication
- Use a specific device to connect

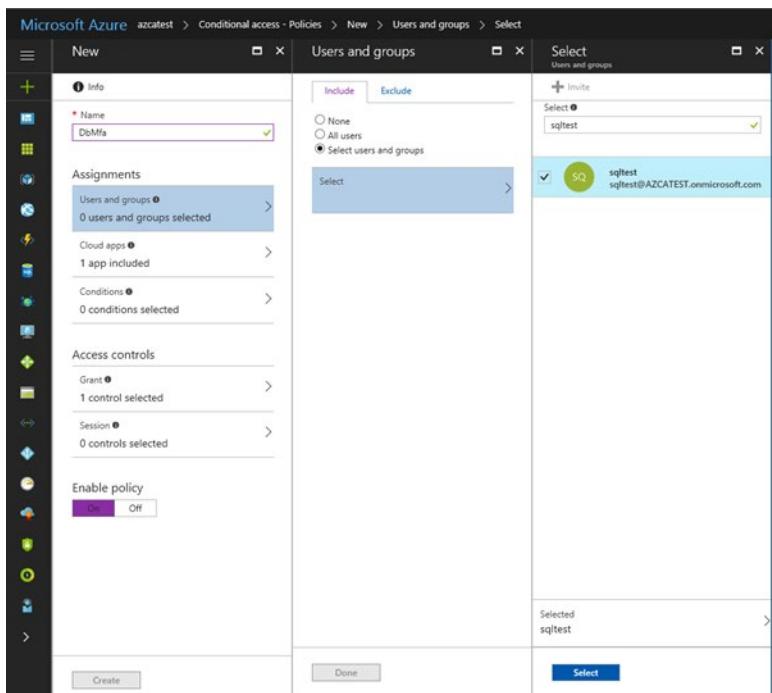
Given the amount of data that could potentially be stored, Azure Synapse Analytics dedicated SQL pools supports conditional access to provide protection for your data. It does require that Azure Synapse Analytics is configured to support Azure Active Directory, and that if you chose multi-factor authentication, that the tool you are using support it.

To configure conditional access, you can perform the following steps:

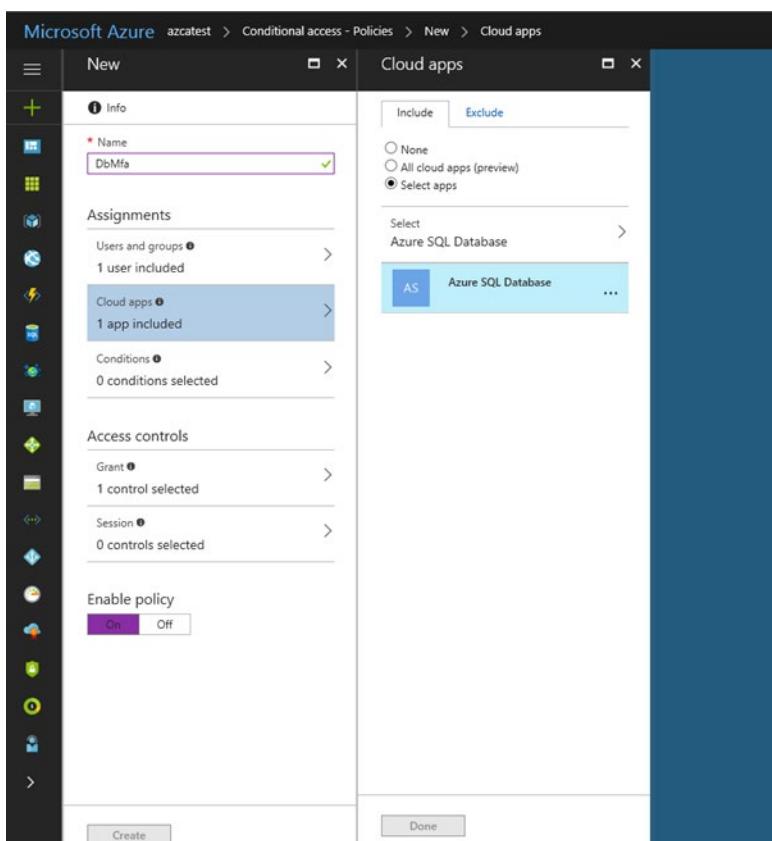
1. Sign in to the Azure portal, select **Azure Active Directory**, and then select **Conditional Access**.

The screenshot shows the Azure Active Directory blade for the directory 'azcatest'. The left sidebar has a 'SECURITY' section with 'Conditional access' highlighted. The main area displays the 'Conditional Access' blade, which includes sections for 'Users and groups', 'Enterprise applications', 'Recommended', and 'App registrations'. A chart titled 'USERS SIGN-INS' shows activity from May 7 to May 28. A 'Quick tasks' sidebar on the right lists options like 'Add a user', 'Sync enabled', and 'Sync with Windows Server AD'.

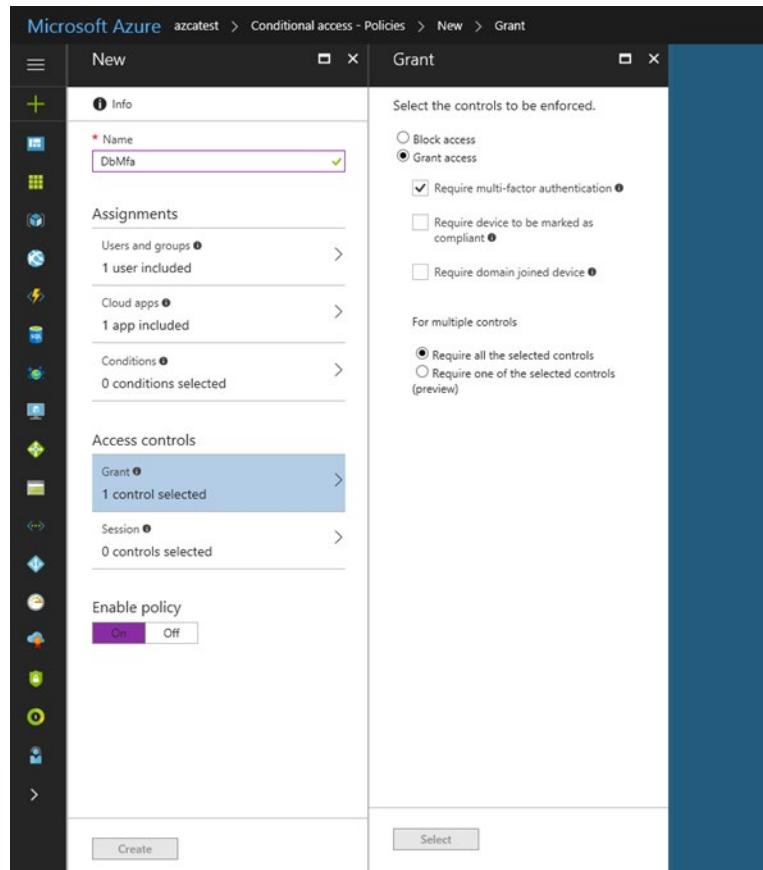
2. In the **Conditional Access-Policies** blade, click **New policy**, provide a name, and then click **Configure rules**.
3. Under **Assignments**, select **Users and groups**, check **Select users and groups**, and then select the user or group for Conditional Access. Click **Select**, and then click **Done** to accept your selection.



4. Select **Cloud apps**, click **Select apps**. You see all apps available for Conditional Access. Select **Azure SQL Database**, at the bottom click **Select**, and then click **Done**.



5. If you can't find **Azure SQL Database** listed in the following third screenshot, complete the following steps:
 - Connect to your database in Azure SQL Database by using SSMS with an Azure AD admin account.
 - Execute CREATE USER [user@yourtenant.com] FROM EXTERNAL PROVIDER.
 - Sign into Azure AD and verify that Azure SQL Database, SQL Managed Instance, or Azure Synapse are listed in the applications in your Azure AD instance.
6. Select **Access controls**, select **Grant**, and then check the policy you want to apply. For this example, we select **Require multi-factor authentication**.



Configure authentication

Authentication is the process of validating credentials as you access resources in a digital infrastructure. This ensures that you can validate that an individual, or a service that wants to access a service in your environment can prove who they are. Azure Synapse Analytics provides several different methods for authentication.

What needs to be authenticated

There are a variety of scenarios that means that authentication must take place to protect the data that is stored in your Azure Synapse Analytics estate.

The common form of authentication is that of individuals who want to access the data in the service. This is typically seen as an individual providing a username and password to authenticate against a service. However, this is also becoming more sophisticated with authentication requests working in combination

with conditional access policies to further secure the authentication process with additional security steps.

What is less obvious is the fact that services must authenticate with other services so that they can operate seamlessly. An example of this is using an Azure Synapse Spark or serverless SQL pool to access data in an Azure Data Lake store. An authentication mechanism must take place in the background to ensure that Azure Synapse Analytics can access the data in the data lake in an authenticated manner.

Finally, there are situations where users and services operate together at the same time. Here you have a combination of both user and service authentication taking place under the hood to ensure that the user is getting access to the data seamlessly. An example of this is using Power BI to view reports in a dashboard that is being serviced by a dedicated SQL pool. Here you have multiple levels of authentication taking place that needs to be managed.

Types of security

The following are the types of authentication that you should be aware of when working with Azure Synapse Analytics.

Azure Active Directory

Azure Active Directory is a directory service that allows you to centrally maintain objects that can be secured. The objects can include user accounts and computer accounts. An employee of an organization will typically have a user account that represents them in the organization's Azure Active Directory tenant, and they then use the user account with a password to authenticate against other resources that are stored within the directory using a process known as single sign-on.

The power of Azure Active Directory is that they only have to log in once, and Azure Active Directory will manage access to other resources based on the information held within it using pass-through authentication. If a user and an instance of Azure Synapse Analytics are part of the same Azure Active Directory, it is possible for the user to access Azure Synapse Analytics without an apparent login. If managed correctly, this process is seamless as the administrator would have given the user authorization to access Azure Synapse Analytics dedicated SQL pool as an example.

In this situation, it is normal for an Azure Administrator to create the user accounts and assign them to the appropriate roles and groups in Azure Active Directory. The Data Engineer will then add the user, or group to which the user belongs to access a dedicated SQL pool.

Managed identities

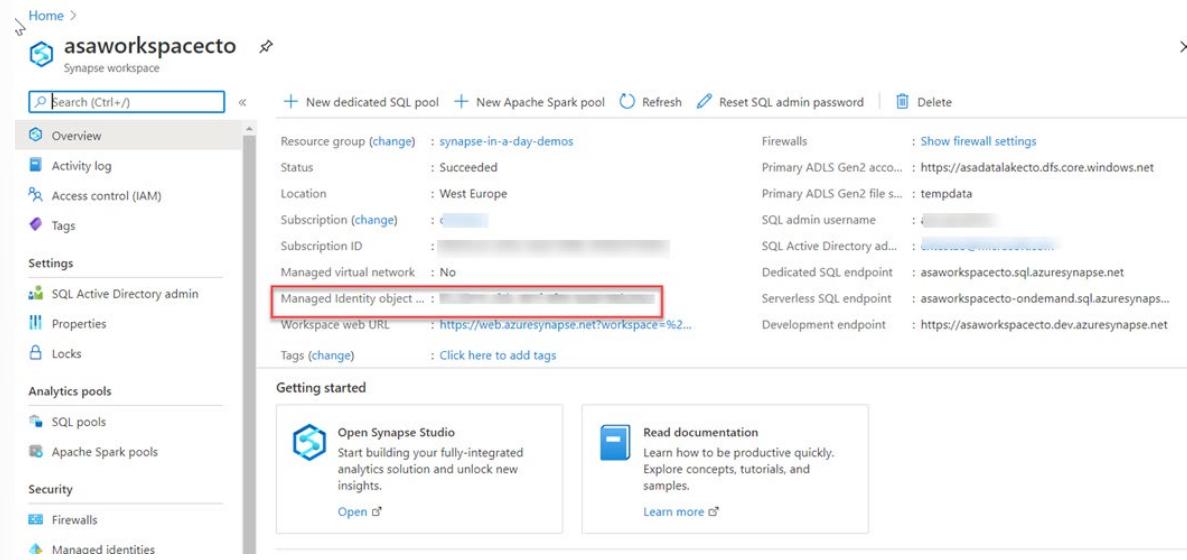
Managed identity for Azure resources is a feature of Azure Active Directory. The feature provides Azure services with an automatically managed identity in Azure AD. You can use the Managed Identity capability to authenticate to any service that supports Azure Active Directory authentication.

Managed identities for Azure resources are the new name for the service formerly known as Managed Service Identity (MSI). A system-assigned managed identity is created for your Azure Synapse workspace when you create the workspace.

Azure Synapse also uses the managed identity to integrate pipelines. The managed identity lifecycle is directly tied to the Azure Synapse workspace. If you delete the Azure Synapse workspace, then the managed identity is also cleaned up.

The workspace managed identity needs permissions to perform operations in the pipelines. You can use the object ID or your Azure Synapse workspace name to find the managed identity when granting permissions.

You can retrieve the managed identity in the Azure portal. Open your Azure Synapse workspace in Azure portal and select **Overview** from the left navigation. The managed identity's object ID is displayed to in the main screen.



The screenshot shows the Azure portal interface for an Azure Synapse workspace. The left sidebar includes links for Home, Overview, Activity log, Access control (IAM), Tags, Settings (SQL Active Directory admin, Properties, Locks), Analytics pools (SQL pools, Apache Spark pools), Security (Firewalls, Managed identities), and Managed identities. The main content area displays workspace details: Resource group (synapse-in-a-day-demos), Status (Succeeded), Location (West Europe), Subscription (change), Subscription ID, Managed virtual network (No), Managed Identity object (highlighted with a red box), Workspace web URL (https://web.azure-synapse.net?workspace=%2...), and Tags (change). Below this, a 'Getting started' section offers links to 'Open Synapse Studio' and 'Read documentation'.

The managed identity information will also show up when you create a linked service that supports managed identity authentication from Azure Synapse Studio.

Launch **Azure Synapse Studio** and select the **Manage** tab from the left navigation. Then select **Linked services** and choose the **+ New** option to create a new linked service.

In the **New linked service** window, type Azure Data Lake Storage Gen2. Select the **Azure Data Lake Storage Gen2** resource type from the list below and choose **Continue**.

In the next window, choose **Managed Identity** for **Authentication method**. You'll see the managed identity's **Name** and **Object ID**.

New linked service (Azure Data Lake Storage Gen2)

Choose a name for your linked service. This name cannot be updated later.

Name *
AzureDataLakeStorage1

Description

Connect via integration runtime *
AutoResolveIntegrationRuntime

Authentication method
Managed Identity

Account selection method
 From Azure subscription Enter manually

Azure subscription
Select all

Storage account name *
[empty input field]

Managed identity name:
Managed identity object ID:

Grant workspace service managed identity access to your Azure Data Lake Storage Gen2.
[Learn more](#)

Test connection
 To linked service To file path

Annotations
+ New

Name

Advanced

Create Back Test connection Cancel

SQL Authentication

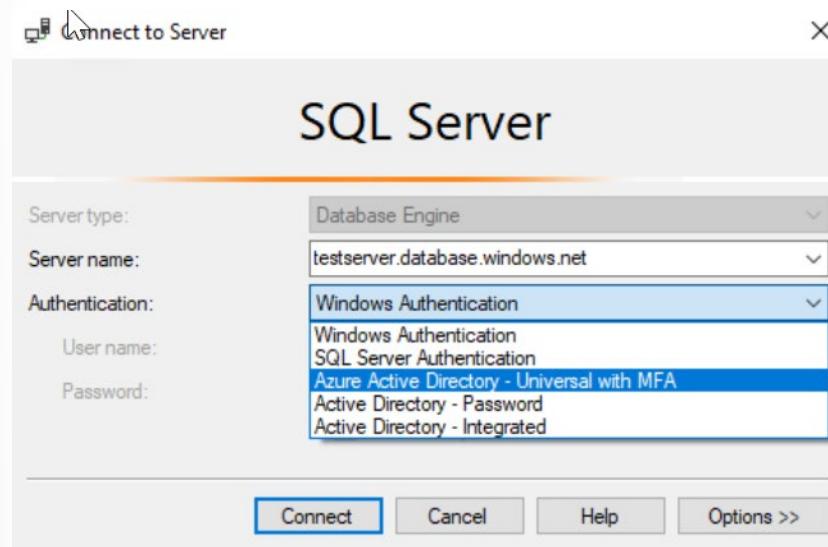
For user accounts that are not part of an Azure Active directory, then using SQL Authentication will be an alternative. In this instance, a user is created in the instance of a dedicated SQL pool. If the user in question requires administrator access, then the details of the user are held in the master database. If

administrator access is not required, you can create a user in a specific database. A user then connects directly to the Azure Synapse Analytics dedicated SQL pool where they are prompted to use a username and password to access the service.

This approach is typically useful for external users who need to access the data, or if you are using third party or legacy applications against the Azure Synapse Analytics dedicated SQL pool

Multi factor authentication

Synapse SQL support connections from SQL Server Management Studio (SSMS) using Active Directory Universal Authentication.



This enables you to operate in environments that use conditional access policies that enforce multi-factor authentication as part of the policy.

Keys

If you are unable to use a managed identity to access resources such as Azure Data Lake then you can use storage account keys and shared access signatures.

With storage account keys. Azure creates two of these keys (primary and secondary) for each storage account you create. The keys give access to everything in the account. You'll find the storage account keys in the Azure portal view of the storage account. Just select **Settings**, and then click **Access keys**.

As a best practice, you shouldn't share storage account keys, and you can use Azure Key Vault to manage and secure the keys.



Azure Key Vault is a secret store: a centralized cloud service for storing app secrets - configuration values like passwords and connection strings that must remain secure at all times. Key Vault helps you control your apps' secrets by keeping them in a single central location and providing secure access, permissions control, and access logging.

The main benefits of using Key Vault are:

- Separation of sensitive app information from other configuration and code, reducing risk of accidental leaks
- Restricted secret access with access policies tailored to the apps and individuals that need them
- Centralized secret storage, allowing required changes to happen in only one place
- Access logging and monitoring to help you understand how and when secrets are accessed

Secrets are stored in individual vaults, which are Azure resources used to group secrets together. Secret access and vault management is accomplished via a REST API, which is also supported by all of the Azure management tools as well as client libraries available for many popular languages. Every vault has a unique URL where its API is hosted.

Shared access signatures

If an external third-party application need access to your data, you'll need to secure their connections without using storage account keys. For untrusted clients, use a shared access signature (SAS). A shared access signature is a string that contains a security token that can be attached to a URI. Use a shared access signature to delegate access to storage objects and specify constraints, such as the permissions and the time range of access. You can give a customer a shared access signature token.

Types of shared access signatures

You can use a service-level shared access signature to allow access to specific resources in a storage account. You'd use this type of shared access signature, for example, to allow an app to retrieve a list of files in a file system or to download a file.

Use an account-level shared access signature to allow access to anything that a service-level shared access signature can allow, plus additional resources and abilities. For example, you can use an account-level shared access signature to allow the ability to create file systems.

Manage authorization through column and row level security

In this topic, we'll go through how you can manage authorization through column and row level security within Azure Synapse Analytics.

We'll start off by talking about column level security in Azure Synapse Analytics, and finish with row level security.

Column level security in Azure Synapse Analytics

Generally speaking, column level security is simplifying a design and coding for the security in your application.

It allows you to restrict column access in order to protect sensitive data.

For example, if you want to ensure that a specific user 'Leo' can only access certain columns of a table because he's in a specific department.

The logic for 'Leo' only to access the columns specified for the department he works in, is a logic that is located in the database tier, rather on the application level data tier.

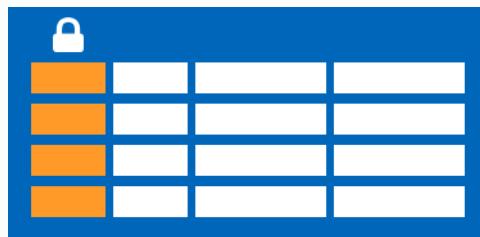
If he needs to access data from any tier, the database should apply the access restriction every time he tries to access data from another tier.

The reason for doing so, is to make sure that your security is reliable and robust since we're reducing the surface area of the overall security system.

Column level security will also eliminate the necessity for the introduction of view, where you would filter out columns, to impose access restrictions on 'Leo'

The way to implement column level security, is by using the GRANT T-SQL statement.

Using this statement, SQL and Azure Active Directory (AAD) support the authentication.



Syntax

The syntax to use for implementing column level security looks as follows:

```
GRANT <permission> [ ,...n ] ON
    [ OBJECT :: ] [ schema_name ]. object_name [ ( column [ ,...n ] ) ] // specifying the column access
        TO <database_principal> [ ,...n ]
        [ WITH GRANT OPTION ]
        [ AS <database_principal> ]
<permission> ::==
    SELECT
    | UPDATE
<database_principal> ::=
    Database_user // specifying the database user
    | Database_role // specifying the database role
    | Database_user_mapped_to_Windows_User
    | Database_user_mapped_to_Windows_Group
```

So when would you use column-level security?

Let's say that you are a financial services firm, and can only have account manager allowed to have access to a customer's social security number, phone numbers or other personal identifiable information. It is imperative to distinguish the role of an account manager versus the manager of the account managers.

Another use case might be related to the Healthcare Industry.

Let's say you have a specific health care provider. This healthcare provider only wants doctors and nurses to be able to access medical records. The billing department should not have access to view this data.

Column-level security might be the option to use.

So how does column level security distinguishes from row-level security.

Let's look into that.

Row level security in Azure Synapse Analytics

Row-level security (RLS) can help you to create a group membership or execution context in order to control not just columns in a database table, but actually, the rows.

RLS, just like column-level security, can simply help and enable your design and coding of your application security.

However, compared to column-level security where it's focused on the columns (parameters), RLS helps you implement restrictions on data row access.

Let's say that your employee can only access rows of data that are important of the department, you should implement RLS.

If you want to restrict for example, customer's data access that is only relevant to the company, you can implement RLS.

The restriction on access of the rows, is a logic that is located in the database tier, rather on the application level data tier.

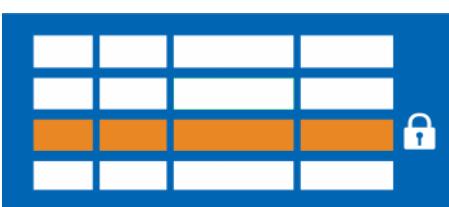
If 'Leo' needs to access data from any tier, the database should apply the access restriction every time he tries to access data from another tier. The reason for doing so, is to make sure that your security is reliable and robust since we're reducing the surface area of the overall security system.

The way to implement RLS is by using the CREATE SECURITY POLICY[!INCLUDEtsql] statement.

The predicates are created as inline table-valued functions.

It is imperative to understand that within Azure Synapse, it only supports filter predicates.

If you need to use a block predicate, you won't be able to find support at this moment within in Azure synapse.



Description of row level security in relation to filter predicates

RLS within Azure Synapse supports one type of security predicates, which are Filter predicates, not block predicates.

What filter predicates do, are silently filtering the rows that are available for read operations such as SELECT, UPDATE, DELETE.

The access to row-level data in a table, is restricted as an inline table-valued function, which is a security predicate.

This table-valued function will then be invoked and enforced by the security policy that you need.

An application, is not aware of rows that are filtered from the result set for filter predicates.

So what will happen is that if all rows are filtered, a null set is returned.

When you are using filter predicates, it will be applied when data is read from the base table.

The filter predicate affects all get operations such as SELECT, DELETE, UPDATE.

You are unable to select or delete rows that have been filtered. It is not possible for you to update a row that has been filtered.

What you can do, is update rows in a way that they will be filtered afterwards.

Use cases

We've already mentioned some use cases for RLS.

Another use case might where you have created a multi-tenant application where you create a policy where logical separations of a tenants data rows from another tenants data rows are enforced.

In order to implement this efficiently, it is highly recommended to store data for many tenants in a single table.

When we look at RLS filter predicates, they are functionally equivalent to appending a **WHERE** clause. The predicate can be as sophisticated as business practices dictate, or the clause can be as simple as `WHERE TenantId = 42`.

When we look at RLS more formally, RLS introduces predicate based access control.

The reason why RLS can be used for predicate access control is that it is flexible, centralized, predicate-based evaluation.

The filter predicate can be based on metadata or any other criteria you would determine as appropriate. The predicate is used as a criterion to determine if the user has the appropriate access to the data based on user attributes.

Label-based access control can be implemented by using predicate-based access control.

Permissions

If you want to create, alter or drop the security policies, you would have to use the **ALTER ANY SECURITY POLICY** permission.

The reason for that is when you are creating or dropping a security policy it requires **ALTER** permissions on the schema.

In addition to that, there are other permissions required for each predicate that you would add:

- **SELECT** and **REFERENCES** permissions on the inline table-valued function being used as a predicate.
- **REFERENCES** permission on the table that you target to be bound to the policy.
- **REFERENCES** permission on every column from the target table used as arguments.

Once you've set up the security policies, they will apply to all the users (including dbo users in the database)

Even though DBO users can alter or drop security policies, their changes to the security policies can be audited.

If you have special circumstances where highly privileged users, like a sysadmin or db_owner, need to see all rows to troubleshoot or validate data, you would still have to write the security policy in order to allow that.

If you have created a security policy where `SCHEMABINDING = OFF`, in order to query the target table, the user must have the **SELECT** or **EXECUTE** permission on the predicate function.

They also need permissions to any additional tables, views, or functions used within the predicate function.

If a security policy is created with `SCHEMABINDING = ON` (the default), then these permission checks are bypassed when users query the target table.

Best practices

There are some best practices to take in mind when you want to implement RLS.

We recommended creating a separate schema for the RLS objects.

RLS objects in this context would be the predicate functions, and security policies.

Why is that a best practice?

It helps to separate the permissions that are required on these special objects from the target tables. In addition to that, separation for different policies and predicate functions may be needed in multi-tenant-databases.

However, it is not a standard for every case.

Another best practice to bear in mind is that the **ALTER ANY SECURITY POLICY** permission should only be intended for highly privileged users (such as a security policy manager).

The security policy manager should not require **SELECT** permission on the tables they protect.

In order to avoid potential runtime errors, you should take in mind type conversions in predicate functions that you write.

Also, you should try to avoid recursion in predicate functions.

The reason for this is to avoid performance degradation.

Even though the query optimizer will try to detect the direct recursions, there is no guarantee to find the indirect recursions.

With an indirect recursion we mean where a second function call the predicate function.

It would also be recommended to avoid the use of excessive table joins in predicate functions. This would maximize performance.

Generally speaking when it comes to the logic of predicates, you should try to avoid logic that depends on session-specific SET options. Even though this is highly unlikely to be used in practical applications, predicate functions whose logic depends on certain session-specific **SET** options can leak information if users are able to execute arbitrary queries. For example, a predicate function that implicitly converts a string to **datetime** could filter different rows based on the **SET DATEFORMAT** option for the current session.

Exercise - Manage authorization through column and row level security

In this exercise, examples are shown how you can manage authorization through column and row level security.

An example of column level security

The following example shows how to restrict `TestUser` from accessing the `SSN` column of the `Membership` table:

Create `Membership` table with `SSN` column used to store social security numbers:

```
CREATE TABLE Membership
(MemberID int IDENTITY,
 FirstName varchar(100) NULL,
 SSN char(9) NOT NULL,
 LastName varchar(100) NOT NULL,
 Phone varchar(12) NULL,
 Email varchar(100) NULL);
```

Allow `TestUser` to access all columns except for the `SSN` column, which has the sensitive data:

```
GRANT SELECT ON Membership(MemberID, FirstName, LastName, Phone, Email) TO
TestUser;
```

Queries executed as TestUser will fail if they include the SSN column:

```
SELECT * FROM Membership;

-- Msg 230, Level 14, State 1, Line 12
-- The SELECT permission was denied on the column 'SSN' of the object
'Membership', database 'CLS_TestDW', schema 'dbo'.
```

An example of row level security

This scenario gives you an example for row level security on an Azure Synapse external table.

This short example creates three users and an external table with six rows. It then creates an inline table-valued function and a security policy for the external table. The example shows how select statements are filtered for the various users.

Prerequisites

- You must have a SQL pool. See Create a Synapse SQL pool
- The server hosting your SQL pool must be registered with AAD and you must have an Azure storage account with Storage Blob Contributor permissions. Follow the steps here.
- Create a file system for your Azure Storage account. Use Storage Explorer to view your storage account. Right click on containers and select *Create file system*.

Once you have the prerequisites in place, create three user accounts that will demonstrate different access capabilities.

```
--run in master
CREATE LOGIN Manager WITH PASSWORD = '<user_password>'
GO
CREATE LOGIN Sales1 WITH PASSWORD = '<user_password>'
GO
CREATE LOGIN Sales2 WITH PASSWORD = '<user_password>'
GO

--run in master and your SQL pool database
CREATE USER Manager FOR LOGIN Manager;
CREATE USER Sales1 FOR LOGIN Sales1;
CREATE USER Sales2 FOR LOGIN Sales2 ;
```

Create a table to hold data.

```
CREATE TABLE Sales
(
    OrderID int,
    SalesRep sysname,
    Product varchar(10),
    Qty int
);
```

Populate the table with six rows of data, showing three orders for each sales representative.

```
INSERT INTO Sales VALUES (1, 'Sales1', 'Valve', 5);
INSERT INTO Sales VALUES (2, 'Sales1', 'Wheel', 2);
INSERT INTO Sales VALUES (3, 'Sales1', 'Valve', 4);
INSERT INTO Sales VALUES (4, 'Sales2', 'Bracket', 2);
INSERT INTO Sales VALUES (5, 'Sales2', 'Wheel', 5);
INSERT INTO Sales VALUES (6, 'Sales2', 'Seat', 5);
-- View the 6 rows in the table
SELECT * FROM Sales;
```

Create an Azure Synapse external table from the Sales table you just created.

```
CREATE MASTER KEY ENCRYPTION BY PASSWORD = '<user_password>';

CREATE DATABASE SCOPED CREDENTIAL msi_cred WITH IDENTITY = 'Managed Service
Identity';

CREATE EXTERNAL DATA SOURCE ext_datasource_with_abfss WITH (TYPE = hadoop,
LOCATION = 'abfss://<file_system_name@storage_account>.dfs.core.windows.
net', CREDENTIAL = msi_cred);

CREATE EXTERNAL FILE FORMAT MSIFormat WITH (FORMAT_TYPE=DELIMITEDTEXT);

CREATE EXTERNAL TABLE Sales_ext WITH (LOCATION='<your_table_name>', DATA_
SOURCE=ext_datasource_with_abfss, FILE_FORMAT=MSIFormat, REJECT_TYPE=Per-
centage, REJECT_SAMPLE_VALUE=100, REJECT_VALUE=100)
AS SELECT * FROM sales;
```

Grant SELECT for the three users on the external table Sales_ext that you created.

```
GRANT SELECT ON Sales_ext TO Sales1;
GRANT SELECT ON Sales_ext TO Sales2;
GRANT SELECT ON Sales_ext TO Manager;
```

Create a new schema, and an inline table-valued function, you may have completed this in example A. The function returns 1 when a row in the SalesRep column is the same as the user executing the query (@SalesRep = USER_NAME()) or if the user executing the query is the Manager user (USER_NAME() = 'Manager').

```
CREATE SCHEMA Security;
GO

CREATE FUNCTION Security.fn_securitypredicate(@SalesRep AS sysname)
RETURNS TABLE
WITH SCHEMABINDING
AS
RETURN SELECT 1 AS fn_securitypredicate_result
WHERE @SalesRep = USER_NAME() OR USER_NAME() = 'Manager';
```

Create a security policy on your external table using the inline table-valued function as a filter predicate. The state must be set to ON to enable the policy.

```
CREATE SECURITY POLICY SalesFilter_ext  
ADD FILTER PREDICATE Security.fn_securitypredicate(SalesRep)  
ON dbo.Sales_ext  
WITH (STATE = ON);
```

Now test the filtering predicate, by selecting from the Sales_ext external table. Sign in as each user, Sales1, Sales2, and manager. Run the following command as each user.

```
SELECT * FROM Sales_ext;
```

The Manager should see all six rows. The Sales1 and Sales2 users should only see their sales.

Alter the security policy to disable the policy.

```
ALTER SECURITY POLICY SalesFilter_ext  
WITH (STATE = OFF);
```

Now the Sales1 and Sales2 users can see all six rows.

Connect to the Azure Synapse database to clean up resources

```
DROP USER Sales1;  
DROP USER Sales2;  
DROP USER Manager;  
  
DROP SECURITY POLICY SalesFilter_ext;  
DROP TABLE Sales;  
DROP EXTERNAL TABLE Sales_ext;  
DROP EXTERNAL DATA SOURCE ext_datasource_with_abfss ;  
DROP EXTERNAL FILE FORMAT MSIFormat;  
DROP DATABASE SCOPED CREDENTIAL msi_cred;  
DROP MASTER KEY;
```

Connect to logical master to clean up resources.

```
DROP LOGIN Sales1;  
DROP LOGIN Sales2;  
DROP LOGIN Manager;
```

Manage sensitive data with Dynamic Data Masking

In this section, we are going to talk about Dynamic Data Masking.

What is Dynamic Data Masking:

Azure SQL Database, Azure SQL Managed Instance, and Azure Synapse Analytics support Dynamic Data Masking.

It's all in the name, Dynamic Data Masking is masking and ensures limited data exposure to non-privileged users, such that they can't see it.

It also helps you in preventing unauthorized access to sensitive data.

The way Dynamic Data Masking does it, is helping customers to designate how much of the sensitive

data to reveal such that it has minimal impact on the application layer.

Dynamic Data Masking is a policy-based security feature.

It will hide the sensitive data in a result set of a query that runs over designated database fields. However, the data in the database will not be changed.

Let's give you an example how it works.

Let's say you work at a bank as a service representative in a call center.

Sometime, due to compliance, the caller has to identify themselves by giving them several digits of their credit card number that they might have an issue with.

However, these data items, should not be fully exposed to the service representative in that call center, answering the call.

If you would define a masking rule, that masks all but the last four digits for example of that credit card number, you would get a query that only gives as a result the last four digits of the credit card number.

If the caller, for example, also had to provide the representative with personal information, that should not be seen by the developer that can query the production environments in order to troubleshoot, you should appropriately mask data in order to protect the given personal data such that compliance is not violated.

For Azure Synapse Analytics, the way to set up a Dynamic Data Masking policy is using PowerShell or the REST API.

Bear in mind that it won't be possible for Azure Synapse Analytics to set the Dynamic Data Masking policy in the Azure portal through selecting the Dynamic Data Masking page under Security in the SQL DB configuration pane.

You need to set it up using PowerShell or REST API as mentioned before.

However, the configuration of the Dynamic Data Masking policy can be done by the Azure SQL Database admin, server admin, or SQL Security Manager roles.

In Azure Synapse Analytics, you can find Dynamic Data Masking here;

The screenshot shows the Azure Synapse Analytics interface for a Dedicated SQL pool named SQLPool01. The left sidebar includes links for Overview, Activity log, Access control (IAM), Tags, Settings (Workload management, Maintenance schedule, Geo-backup policy, Connection strings, Properties, Locks), Security (Auditing, Data Discovery & Classification, Dynamic Data Masking, Security Center, Transparent data encryption). The 'Dynamic Data Masking' link is highlighted with a red box. The main content area shows a 'Masking rules' section with a table for 'Mask name' and 'Mask Function'. It notes that no masking rules have been created. Below this is a section for 'SQL users excluded from masking' with a note that administrators are always excluded, and a specific row for 'SQL users excluded from masking (administrators are always excluded)' with a checked status. A 'Learn more - Getting Started Guide' link is also present. A table titled 'Recommended fields to mask' lists various columns across different schemas and tables, each with an 'Add mask' button.

Looking into Dynamic Data Masking Policies:

- **SQL users are excluded from masking**

A couple of SQL users or Azure AD identities can get unmasked data in the SQL query results. Users with administrator privileges are always excluded from masking, and see the original data without any mask.

- **Masking rules** - Masking rules are a set of rules that define the designated fields to be masked including the masking function that is used. The designated fields can be defined using a database schema name, table name, and column name.
- **Masking functions** - Masking functions are a set of methods that control the exposure of data for different scenarios.

Set up Dynamic Data Masking for your database in Azure Synapse Analytics using PowerShell cmdlets

In this part, we are going to look into Dynamic Data Masking for a database in Azure Synapse Analytics using PowerShell cmdlets.

- Data masking policies
- Get-AzSqlDatabaseDataMaskingPolicy

The Get-AzSqlDatabaseDataMaskingPolicy gets the data masking policy for a database.

The syntax for the Get-AzSqlDatabaseDataMaskingPolicy in PowerShell is as follows:

```
Get-AzSqlDatabaseDataMaskingPolicy [-ServerName] <String> [-DatabaseName]
<String>
[-ResourceGroupName] <String> [-DefaultProfile <IAzureContextContainer>]
[-WhatIf] [-Confirm]
[<CommonParameters>]
```

What the **Get-AzSqlDatabaseDataMaskingPolicy** cmdlet does, is getting the data masking policy of an Azure SQL database.

To use this cmdlet in PowerShell, you'd have to specify the following parameters to identify the database:

- *ResourceGroupName*: name of the resource group you deployed the database in
- *ServerName*: sql server name
- *DatabaseName* : name of the database

This cmdlet is also supported by the SQL Server Stretch Database service on Azure.

- Set-AzSqlDatabaseDataMaskingPolicy

The Set-AzSqlDatabaseDataMaskingPolicy sets data masking for a database.

The syntax for the Set-AzSqlDatabaseDataMaskingPolicy in PowerShell is as follows:

```
Set-AzSqlDatabaseDataMaskingPolicy [-PassThru] [-PrivilegedUsers <String>]
[-DataMaskingState <String>]
[-ServerName] <String> [-DatabaseName] <String> [-ResourceGroupName]
<String>
[-DefaultProfile <IAzureContextContainer>] [-WhatIf] [-Confirm] [<CommonPa-
rameters>]
```

What the **Set-AzSqlDatabaseDataMaskingPolicy** cmdlet does is setting the data masking policy for an Azure SQL database.

To use this cmdlet in PowerShell, you'd have to specify the following parameters to identify the database:

- *ResourceGroupName*: name of the resource group that you deployed the database in
- *ServerName* : sql server name
- *DatabaseName* : name of the database

In addition, you will need to set the *DataMaskingState* parameter to specify whether data masking operations are enabled or disabled.

If the cmdlet succeeds and the *PassThru* parameter is used, it will return an object describing the current data masking policy in addition to the database identifiers.

Database identifiers can include, **ResourceGroupName**, **ServerName**, and **DatabaseName**.

This cmdlet is also supported by the SQL Server Stretch Database service on Azure.

- Data masking rules
- Get-AzSqlDatabaseDataMaskingRule

The Get-AzSqlDatabaseDataMaskingRule Gets the data masking rules from a database.

The syntax for the Get-AzSqlDatabaseDataMaskingRule in PowerShell is as follows:

```
Get-AzSqlDatabaseDataMaskingRule [-SchemaName <String>] [-TableName
<String>] [-ColumnName <String>]
```

```
[-ServerName] <String> [-DatabaseName] <String> [-ResourceGroupName]  
<String>  
[-DefaultProfile <IAzureContextContainer>] [-WhatIf] [-Confirm] [<CommonPa-  
rameters>]
```

What the **Get-AzSqlDatabaseDataMaskingRule** cmdlet does is getting either a specific data masking rule or all of the data masking rules for an Azure SQL database.

To use the cmdlet in PowerShell, you'd have to specify the following parameters to identify the database:

To use this cmdlet in PowerShell, you'd have to specify the following parameters to identify the database:

- *ResourceGroupName*: name of the resource group that you deployed the database in
- *ServerName* : sql server name
- *DatabaseName* : name of the database

You'd also have to specify the *RuleId* parameter to specify which rule this cmdlet returns.

If you do not provide *RuleId*, all the data masking rules for that Azure SQL database are returned.

This cmdlet is also supported by the SQL Server Stretch Database service on Azure.

- New-AzSqlDatabaseDataMaskingRule

The New-AzSqlDatabaseDataMaskingRule creates a data masking rule for a database.

The syntax for the New-AzSqlDatabaseDataMaskingRule in PowerShell is as follows:

```
New-AzSqlDatabaseDataMaskingRule -MaskingFunction <String> [-PrefixSize  
 <UInt32>] [-ReplacementString <String>]  
 [-SuffixSize <UInt32>] [-NumberFrom <Double>] [-NumberTo <Double>] [-PassThru]  
 [-SchemaName <String>]  
 [-TableName <String>] [-ColumnName <String>] [-ServerName] <String> [-Data-  
 baseName] <String>  
 [-ResourceGroupName] <String> [-DefaultProfile <IAzureContextContainer>]  
 [-WhatIf] [-Confirm]  
 [<CommonParameters>]
```

What the **New-AzSqlDatabaseDataMaskingRule** cmdlet does is creating a data masking rule for an Azure SQL database.

To use this cmdlet in PowerShell, you'd have to specify the following parameters to identify the rule:

- *ResourceGroupName*: name of the resource group that you deployed the database in
- *ServerName* : sql server name
- *DatabaseName* : name of the database

Providing the *TableName* and *ColumnName* is necessary in order to specify the target of the rule.

The *MaskingFunction* parameter is necessary to define how the data is masked.

If *MaskingFunction* has a value of Number or Text, you can specify the *NumberFrom* and *NumberTo* parameters, for number masking, or the *PrefixSize*, *ReplacementString*, and *SuffixSize* for text masking.

If the command succeeds and the *PassThru* parameter is used, the cmdlet returns an object describing the data masking rule properties in addition to the rule identifiers.

Rule identifiers can be, for example, *ResourceGroupName*, *ServerName*, *DatabaseName*, and *RuleID*.

This cmdlet is also supported by the SQL Server Stretch Database service on Azure.

- Remove-AzSqlDatabaseDataMaskingRule

The Remove-AzSqlDatabaseDataMaskingRule removes a data masking rule from a database.

The syntax for the Remove-AzSqlDatabaseDataMaskingRule in PowerShell is as follows:

```
Remove-AzSqlDatabaseDataMaskingRule [-PassThru] [-Force] -SchemaName <String> -TableName <String> -ColumnName <String> [-ServerName] <String> [-DatabaseName] <String> [-ResourceGroupName] <String> [-DefaultProfile <IAzureContextContainer>] [-WhatIf] [-Confirm] [<CommonParameters>]
```

What the **Remove-AzSqlDatabaseDataMaskingRule** cmdlet does, is it removes a specific data masking rule from an Azure SQL database.

To use this cmdlet in PowerShell, you'd have to specify the following parameters to identify the rule that needs to be removed:

- *ResourceGroupName*: name of the resource group that you deployed the database in
- *ServerName* : sql server name
- *DatabaseName* : name of the database
- *RuleId* : identifier of the rule

This cmdlet is also supported by the SQL Server Stretch Database service on Azure.

- Set-AzSqlDatabaseDataMaskingRule

The Set-AzSqlDatabaseDataMaskingRule Sets the properties of a data masking rule for a database.

The syntax for the Set-AzSqlDatabaseDataMaskingRule in PowerShell is as follows:

```
Set-AzSqlDatabaseDataMaskingRule [-MaskingFunction <String>] [-PrefixSize <UInt32>] [-ReplacementString <String>] [-SuffixSize <UInt32>] [-NumberFrom <Double>] [-NumberTo <Double>] [-PassThru] -SchemaName <String> -TableName <String> -ColumnName <String> [-ServerName] <String> [-DatabaseName] <String> [-ResourceGroupName] <String> [-DefaultProfile <IAzureContextContainer>] [-WhatIf] [-Confirm] [<CommonParameters>]
```

What the **Set-AzSqlDatabaseDataMaskingRule** cmdlet does is setting a data masking rule for an Azure SQL database.

To use this cmdlet in PowerShell, you'd have to specify the following parameters to identify the rule:

- *ResourceGroupName*: name of the resource group that you deployed the database in
- *ServerName* : sql server name
- *DatabaseName* : name of the database
- *RuleId* : identifier of the rule

You can provide any of the parameters of *SchemaName*, *TableName*, and *ColumnName* to retarget the rule.

Specify the *MaskingFunction* parameter to modify how the data is masked.

If you specify a value of Number or Text for *MaskingFunction*, you can specify the *NumberFrom* and *NumberTo* parameters for number masking or the *PrefixSize*, *ReplacementString*, and *SuffixSize* parameters for text masking.

If the command succeeds, and if you specify the *PassThru* parameter, the cmdlet returns an object that describes the data masking rule properties and the rule identifiers.

Rule identifiers can be, **ResourceGroupName**, **ServerName**, **DatabaseName**, and **RuleId**.

This cmdlet is also supported by the SQL Server Stretch Database service on Azure.

Set up Dynamic Data Masking for your database in Azure Synapse Analytics using the REST API

For setting up Dynamic Data Masking in Azure Synapse Analytics, the other possibility is make use of the REST API.

It will enable to programmatically manage data masking policy and rules.

The REST API will support the following operations:

- Data masking policies
- Create Or Update

The Create Or Update masking policy using the REST API will create or update a database data masking policy.

In HTTP the following request can be made:

```
PUT https://management.azure.com/subscriptions/{subscriptionId}/resourceGroups/{resourceGroupName}/providers/Microsoft.Sql/servers/{serverName}/databases/{databaseName}/dataMaskingPolicies/Default?api-version=2014-04-01
```

The following parameters need to be passed through:

- *SubscriptionID*: the ID of the subscription
- *ResourceGroupName*: name of the resource group that you deployed the database in
- *ServerName* : sql server name
- *DatabaseName* : name of the database
- *dataMaskingPolicyName*: the name of the data masking policy
- *api version*: version of the api that is used.
- Get

The Get policy, Gets a database data masking policy.

In HTTP the following request can be made:

```
GET https://management.azure.com/subscriptions/{subscriptionId}/resourceGroups/{resourceGroupName}/providers/Microsoft.Sql/servers/{serverName}/databases/{databaseName}/dataMaskingPolicies/Default?api-version=2014-04-01
```

The following parameters need to be passed through:

- *SubscriptionID*: the ID of the subscription
- *ResourceGroupName*: name of the resource group that you deployed the database in
- *ServerName* : sql server name
- *DatabaseName* : name of the database
- *dataMaskingPolicyName*: the name of the data masking policy
- *api version*: version of the api that is used.
- Data masking rules
- Create Or Update

The Create or Update masking rule creates or updates a database data masking rule.

In HTTP the following request can be made:

```
PUT https://management.azure.com/subscriptions/{subscriptionId}/resourceGroups/{resourceGroupName}/providers/Microsoft.Sql/servers/{serverName}/databases/{databaseName}/dataMaskingPolicies/Default/rules/{dataMaskingRuleName}?api-version=2014-04-01
```

The following parameters need to be passed through:

- *SubscriptionID*: the ID of the subscription
- *ResourceGroupName*: name of the resource group that you deployed the database in
- *ServerName* : sql server name
- *DatabaseName* : name of the database
- *dataMaskingPolicyName*: the name of the data masking policy
- *dataMaskingRuleName*: the name of the rule for data masking
- *api version*: version of the api that is used.
- List By Database

The List By Database request gets a list of database data masking rules.

In HTTP the following request can be made:

```
GET https://management.azure.com/subscriptions/{subscriptionId}/resourceGroups/{resourceGroupName}/providers/Microsoft.Sql/servers/{serverName}/databases/{databaseName}/dataMaskingPolicies/Default/rules?api-version=2014-04-01
```

The following parameters need to be passed through:

- *SubscriptionID*: the ID of the subscription
- *ResourceGroupName*: name of the resource group that you deployed the database in
- *ServerName* : sql server name
- *DatabaseName* : name of the database
- *dataMaskingPolicyName*: the name of the data masking policy

- *api version*: version of the api that is used.

Implement encryption in Azure Synapse Analytics

In this section, we will go through Transparent Data Encryption and TokenLibrary for Apache Spark.

What is transparent data encryption

Transparent data encryption (TDE) is an encryption mechanism to help you protect Azure Synapse Analytics.

It will protect Azure Synapse Analytics against threats of malicious offline activity.

The way TDE will do so, is by encrypting data at rest.

TDE performs real-time encryption as well as decryption of the database, associated backups, and transaction log files at rest without you having to make changes to the application.

In order to use TDE for Azure Synapse Analytics, you will have to manually enable it.

What TDE does is performing I/O encryption and decryption of data at the page level in real time.

When a page is read into memory, it is decrypted. It is encrypted before writing it to disk.

TDE encrypts the entire data base storage, using a symmetric key called a Database Encryption Key (DEK). When you start up a database, the encrypted Database Encryption Key is decrypted when it then will be used for decryption and re-encryption of the database files in the SQL Server database engine.

The DEK is protected by the Transparent Data Encryption Protector.

This protector can be either a service-managed certificate, which is referred to as service-managed transparent data encryption, or an asymmetric key that is stored in Azure Key Vault (customer-managed transparent data encryption).

What is important to understand is that for Azure Synapse Analytics, this TDE protector is set on the server level.

There it is inherited by all the databases that are attached or aligned to that server.

The term server refers both to server and instance.

Service-managed transparent data encryption

As stated above, the DEK that is protected by the Transparent Encryption protector can be service-managed certificate which we call service-managed TDE.

When you look in Azure, that default setting means that the DEK is protected by a built-in certificate unique for each server with encryption algorithm AES256.

When a database is in a geo-replicated relationship then primary and the geo-secondary database are protected by the primary database's parent server key.

If the databases are connected to the same server, they will also have the same built-in AES 256 certificate.

As Microsoft we automatically rotate the certificates in compliance with the internal security policy.

The root key is protected by a Microsoft internal secret store.

Microsoft also seamlessly moves and manages the keys as needed for geo-replication and restores.

Transparent data encryption with bring your own key for customer-managed transparent data encryption

As stated above, the DEK that is protected by the Transparent Data Encryption Protector can also be customer managed by bringing an asymmetric key that is stored in Azure Key Vault (customer-managed

transparent data encryption).

This is also referred to as Bring Your Own Key (BYOK) support for TDE.

When this is the scenario that is applicable to you, the TDE Protector that encrypts the DEK is a customer-managed asymmetric key.

This is stored in your own and managed Azure Key Vault.

Azure Key Vault is Azure's cloud-based external key management system.

This managed key never leaves the key vault.

The TDE Protector can be generated by the key vault.

Another option is to transfer the TDE Protector to the key vault from, for example, an on-premise hardware security module (HSM) device.

Azure Synapse Analytics needs to be granted permissions to the customer-owned key vault in order to decrypt and encrypt the DEK.

If permissions of the server to the key vault are revoked, a database will be inaccessible, and all data is encrypted.

By using Azure Key Vault integration for TDE, you have control over the key management tasks such as key rotations, key backups, and key permissions.

It also enables you for auditing and reporting on all the TDE protectors when using the Azure Key Vault functionality.

The reason for using Key Vault is that it provides you with a central key management system where tightly monitored HSMs are leveraged.

It also enables you to separate duties of management of keys and data in order to meet compliance with security policies.

Manage transparent data encryption in the Azure portal.

For Azure Synapse Analytics, you can manage TDE for the database in the Azure portal after you've signed in with the Azure Administrator or Contributor account.

The TDE settings can be found under your user database.

Home > SQLPool01 (asaworkspacecto/SQLPool01)

The screenshot shows the Azure portal interface for managing a Dedicated SQL pool named "SQLPool01". The left sidebar contains navigation links for Overview, Activity log, Access control (IAM), Tags, Settings (Workload management, Maintenance schedule, Geo-backup policy, Connection strings, Properties, Locks), Security (Auditing, Data Discovery & Classification, Dynamic Data Masking, Security Center), and Common Tasks (Open in Visual Studio). The "Transparent data encryption" link is highlighted with a red box. The main content area displays information about TDE, stating it encrypts databases, backups, and logs at rest without changes to the application. It shows the current status as "Unencrypted" and provides a "Data encryption" toggle switch, which is currently set to "OFF" (highlighted with a red box). A "Learn more" link is also present.

It is by default that the service-managed TDE is used and therefore a TDE certificate is automatically generated for the server that contains that database.

Moving a transparent data encryption protected database

In some use cases you need to move a database that is protected with TDE.

Within Azure, there is no need to decrypt the databases.

The TDE settings on the source database or primary database, will be inherited on the target.

Some of the operations within Azure that inherited the TDE are:

- Geo-restore
- Self-service point-in-time restore
- Restoration of a deleted database
- Active geo-replication
- Creation of a database copy
- Restore of backup file to Azure SQL Managed Instance

If you export a TDE-protected database, the exported content is not encrypted. This will be stored in an unencrypted BACPAC file. You need to make sure that you protect this BACPAC file and enable TDE as soon as the import of the bacpac file in the new database is finished.

Securing your credentials through linked services with TokenLibrary for Apache Spark

It is quite a common pattern to access data from external sources. Unless the external data source allows anonymous access, it is highly likely that you need to secure your connection with a credential, secret, or connection string.

Within Azure Synapse Analytics, the integration process is simplified by providing linked services. Doing so, the connection details can be stored in the linked service or an Azure Key Vault. If the Linked Service is created, Apache spark can reference the linked service to apply the connection information in your code.

When you want to access files from the Azure Data Lake Storage Gen 2 within your Azure Synapse Analytics Workspace, it uses AAD passthrough for the authentication.

Therefore, there is no need to use the TokenLibrary.

However, to connect to other linked services, you are enabled to make a direct call to the TokenLibrary.

An example can be found below:

In order to connect to other linked services, you are enabled to make a direct call to the TokenLibrary by retrieving the connection string.

In order to retrieve the connection string, use the **getConnectionString** function and pass in the **linked service name**.

```
// Scala  
// retrieve connectionstring from TokenLibrary  
  
import com.microsoft.azure.synapse.tokenlibrary.TokenLibrary  
  
val connectionString: String = TokenLibrary.getConnectionString("<LINKED  
SERVICE NAME>")  
println(connectionString)  
  
# Python  
# retrieve connectionstring from TokenLibrary  
  
from pyspark.sql import SparkSession  
  
sc = SparkSession.builder.getOrCreate()  
token_library = sc._jvm.com.microsoft.azure.synapse.tokenlibrary.TokenLi-  
brary  
connection_string = token_library.getConnectionString("<LINKED SERVICE  
NAME>")  
print(connection_string)
```

If you want to Get the connection string as map and parse specific values from a key pair in the connection string, you can find an example below:

To parse specific values from a *key=value* pair in the connection string such as

DefaultEndpointsProtocol=https;AccountName=<AccountName>;AccountKey=<AccountKey>

use the **getConnectionStringAsMap** function and pass the key to return the value.

```
// Linked services can be used for storing and retrieving credentials (e.g,
account key)
// Example connection string (for storage): "DefaultEndpointsProto-
col=https;AccountName=<accountname>;AccountKey=<accountkey>"
import com.microsoft.azure.synapse.tokenlibrary.TokenLibrary

val accountKey: String = TokenLibrary.getConnectionStringAsMap("<LINKED
SERVICE NAME>").get("<KEY NAME>")
println(accountKey)

# Linked services can be used for storing and retrieving credentials (e.g,
account key)
# Example connection string (for storage): "DefaultEndpointsProto-
col=https;AccountName=<accountname>;AccountKey=<accountkey>"
from pyspark.sql import SparkSession

sc = SparkSession.builder.getOrCreate()
token_library = sc._jvm.com.microsoft.azure.synapse.tokenlibrary.TokenLi-
brary
accountKey = token_library.getConnectionStringAsMap("<LINKED SERVICE
NAME>").get("<KEY NAME>")
print(accountKey)
```

Knowledge check

Question 1

You want to configure a private endpoint. You open up Azure Synapse Studio, go to the manage hub, and see that the private endpoints is greyed out. Why is the option not available?

- Azure Synapse Studio does not support the creation of private endpoints.
- A conditional access policy has to be defined first.
- A managed virtual network has not been created.

Question 2

You require you Azure Synapse Analytics Workspace to access an Azure Data Lake Store using the benefits of the security provided by Azure Active Directory. What is the best authentication method to use?

- Storage account keys.
- Shared access signatures.
- Managed identities.

Summary

In this module, you have learned how to approach and implement security to protect your data with Azure Synapse Analytics.

In this module, you have:

- Understood network security options for Azure Synapse Analytics
- Configured conditional access
- Configured Authentication
- Managed authorization through column and row level security
- Managed sensitive data with Dynamic Data masking
- Implemented encryption in Azure Synapse Analytics
- Understood advanced data security options for Azure Synapse Analytics

Configure and manage secrets in Azure Key Vault

Introduction

PetDash is an online pet food delivery company that provides store-to-door service for all their customer's pet needs. They take online orders, store credit cards and personal details in their SQL database, and have a secure website running on Azure App Service to interact with customers. They've been in business a little over a year and Steve, one of the website admins, noticed that their website certificate for the **petdash.com** domain has expired. Steve quickly renews the certificate and gets it installed on the server, and begins to explore ways to ensure that this problem never happens again. The investigation reveals that Azure Key Vault supports certificate management. Even better, Key Vault can communicate with App Service to provide the certification *and* renew it automatically if necessary.

Azure Key Vault helps safeguard cryptographic keys and secrets that cloud applications and services use. Key Vault streamlines the key management process and enables you to maintain control of keys that access and encrypt your data. Developers can create keys for development and testing in minutes, and then migrate them to production keys. Security administrators can grant (and revoke) permission to keys, as needed.

Learning objectives

In this module, you will:

- Explore proper usage of Azure Key Vault
- Manage access to an Azure Key Vault
- Explore certificate management with Azure Key Vault
- Configure a Hardware Security Module Key-generation solution

Guidelines for using Azure Key Vault

Azure Key Vault is a centralized cloud service for storing application secrets such as encryption keys, certificates, and server-side tokens. Key Vault helps you control your applications' secrets by keeping them in a single central location and providing secure access, permissions control, and access logging.

There are three primary concepts used in an Azure Key Vault: *vaults*, *keys*, and *secrets*.

Vaults

You use Azure Key Vault to create multiple secure containers, called vaults. Vaults help reduce the chances of accidental loss of security information by centralizing application secrets storage. Organizations will have several key vaults. Each key vault is a collection of cryptographic keys and cryptographically protected data (call them "secrets") managed by one or more responsible individuals within your organization. These key vaults represent the logical groups of keys and secrets for your organization; those that you want to manage together. They are like folders in the file system. Key vaults also control and log the access to anything stored in them.

You can create and manage vaults using command line tools such as Azure PowerShell or the Azure CLI, using the REST API, or through the Azure portal.

For example, here's a sample Azure CLI command line to create a new vault in a resource group:

```
az keyvault create \
    --resource-group <resource-group> \
    --name <your-unique-vault-name>
```

Here's the same command using Azure PowerShell:

```
New-AzKeyVault -Name <your-unique-vault-name> -ResourceGroupName <re-
source-group>
```

Keys

Keys are the central actor in the Azure Key Vault service. A given key in a key vault is a cryptographic asset destined for a particular use such as the asymmetric master key of Microsoft Azure RMS, or the asymmetric keys used for SQL Server TDE (Transparent Data Encryption), CLE (Column Level Encryption) and Encrypted backup.

Microsoft and your apps don't have access to the stored keys directly once a key is created or added to a key vault. Applications must use your keys by calling cryptography methods on the Key Vault service. The Key Vault service performs the requested operation within its hardened boundary. The application never has direct access to the keys.

Keys can be single instanced (only one key exists) or be versioned. In the versioned case, a key is an object with a primary (active) key and a collection of one or more secondary (archived) keys created when keys are rolled (renewed). Key Vault supports asymmetric keys (RSA 2048). Your applications may use these for encryption or digital signatures.

There are two variations on keys in Key Vault: hardware-protected, and software-protected.

Hardware protected keys

The Key Vault service supports using HSMs that provide a hardened, tamper-resistant environment for cryptographic processing and key generation. Azure has dedicated HSMs validated to FIPS 140-2 Level 2 that Key Vault uses to generate or store keys. These HSM-backed keys are always locked to the boundary of the HSM. When you ask the Key Vault service to decrypt or sign with a key, the operation is performed inside an HSM.

You can import keys from your own hardware security modules (HSMs) and transfer them to Key Vault without leaving the HSM boundary. This scenario is often referred to as *bring your own key*, or BYOK. More details on generating your own HSM-protected key and then transferring it to Azure Key Vault is available in the summary of this module. You can also use these Azure HSMs directly through the Microsoft Azure Dedicated Hardware Security Module (HSM) service if you need to migrate HSM-protected apps or maintain a high security compliance requirement.

Software protected keys

Key Vault can also generate and protect keys using software-based RSA and ECC algorithms. In general, software-protected keys offer most of the features as HSM-protected keys except the FIPS 140-2 Level 2 assurance:

- Your key is still isolated from the application (and Microsoft) in a container that you manage
- It's stored *at rest* encrypted with HSMs
- You can monitor usage using Key Vault logs

The primary difference (besides price) with a software-protected key is when cryptographic operations are performed, they are done in software using Azure compute services while for HSM-protected keys the cryptographic operations are performed within the HSM.

TIP: For production use, it's recommended to use HSM-protected keys and use software-protected keys in only test/pilot scenarios. There is an additional charge for HSM-backed keys per-month if the key is used in that month. The summary page has a link to the pricing details for Azure Key Vault.

You determine the key generation type when you create the key. For example, the Azure PowerShell command `Add-AzureKeyVaultKey` has a `Destination` parameter that can be set to either Software or HSM:

```
$key = Add-AzureKeyVaultKey -VaultName 'contoso' -Name 'MyFirstKey' -Destination 'HSM'
```

Secrets

Secrets are small (less than 10K) data blobs protected by a HSM-generated key created with the Key Vault. Secrets exist to simplify the process of persisting sensitive settings that almost every application has: storage account keys, .PFX files, SQL connection strings, data encryption keys, etc.

Key vault uses

With these three elements, an Azure Key Vault helps address the following issues:

- **Secrets management.** Azure Key Vault can securely store (with HSMs) and tightly control access to tokens, passwords, certificates, API keys, and other secrets.
- **Key management.** Azure Key Vault is a cloud-based key management solution, making it easier to create and control the encryption keys used to encrypt your data. Azure services such as App Service integrate directly with Azure Key Vault and can decrypt secrets without knowledge of the encryption keys.
- **Certificate management.** Azure Key Vault is also a service that lets you easily provision, manage, and deploy public and private SSL/TLS certificates for use with Azure and your internal connected resources. It can also request and renew TLS certificates through partnerships with certificate authorities, providing a robust solution for certificate lifecycle management.

IMPORTANT: Key Vault is designed to store configuration secrets for server applications. It's not intended for storing data belonging to your app's users, and it shouldn't be used in the client-side part of an app. This is reflected in its performance characteristics, API, and cost model.

User data should be stored elsewhere, such as in an Azure SQL database with Transparent Data Encryption, or a storage account with Storage Service Encryption. Secrets used by your application to access those data stores can be kept in Key Vault.

Best practices

Here are some security best practices for using Azure Key Vault.

Best practice	Solution
Grant access to users, groups, and applications at a specific scope.	Use RBAC's predefined roles. For example, to grant access to a user to manage key vaults, you would assign the predefined role Key Vault Contributor to this user at a specific scope. The scope, in this case, would be a subscription, a resource group, or just a specific key vault. If the predefined roles don't fit your needs, you can define your own roles.
Control what users have access to.	Access to a key vault is controlled through two separate interfaces: management plane, and data plane. The management plane and data plane access controls work independently. Use RBAC to control what users have access to. For example, if you want to grant an application the rights to use keys in a key vault, you only need to grant data plane access permissions using key vault access policies. No management plane access is needed for this application. Conversely, if you want a user to be able to read vault properties and tags but not have any access to keys, secrets, or certificates, by using RBAC, you can grant read access to the management plane. No access to the data plane is required.
Store certificates in your key vault.	Azure Resource Manager can securely deploy certificates stored in Azure Key Vault to Azure VMs when the VMs are deployed. By setting appropriate access policies for the key vault, you also control who gets access to your certificate. Another benefit is that you manage all your certificates in one place in Azure Key Vault.
Ensure that you can recover a deletion of key vaults or key vault objects.	Deletion of key vaults or key vault objects can be either inadvertent or malicious. Enable the soft delete and purge protection features of Key Vault, particularly for keys that are used to encrypt data at rest. Deletion of these keys is equivalent to data loss, so you can recover deleted vaults and vault objects if needed. Practice Key Vault recovery operations regularly.

NOTE: If a user has contributor permissions (RBAC) to a key vault management plane, they can grant themselves access to the data plane by setting a key vault access policy. It's recommended that you tightly control who has contributor access to your key vaults, to ensure that only authorized persons can access and manage your key vaults, keys, secrets, and certificates.

Manage access to secrets, certificates, and keys

Key Vault access has two facets: the management of the Key Vault itself, and accessing the data contained in the Key Vault. Documentation refers to these facets as the *management plane* and the *data plane*.

These two areas are separated because the creation of the Key Vault (a management operation) is a different role than storing and retrieving a secret stored in the Key Vault. To access a key vault, all users or apps must have proper *authentication* to identify the caller, and *authorization* to determine the operations the caller can perform.

Authentication

Azure Key Vault uses Azure Active Directory (Azure AD) to authenticate users and apps that try to access a vault. Authentication is always performed by associating the Azure AD tenant of the subscription that the Key Vault is part of, and every user or app making a request having to be known to Azure AD. There is no support for anonymous access to a Key Vault.

Authorization

Management operations (creating a new Azure Key Vault) use role-based access control (RBAC). There is a built-in role **Key Vault Contributor** that provides access to management features of key vaults, but doesn't allow access to the key vault data. This is the recommended role to use. There's also a **Contributor** role that includes full administration rights - including the ability to grant access to the data plane.

Reading and writing data in the Key Vault uses a separate Key Vault *access policy*. A Key Vault access policy is a permission set assigned to a user or managed identity to read, write, and/or delete secrets and keys. You can create an access policy using the CLI, REST API, or Azure portal as shown below.

Add access policy

Add access policy

Configure from template (optional)

Key, Secret, & Certificate Management

Key permissions

9 selected

Secret permissions

7 selected

Certificate permissions

15 selected

Select principal

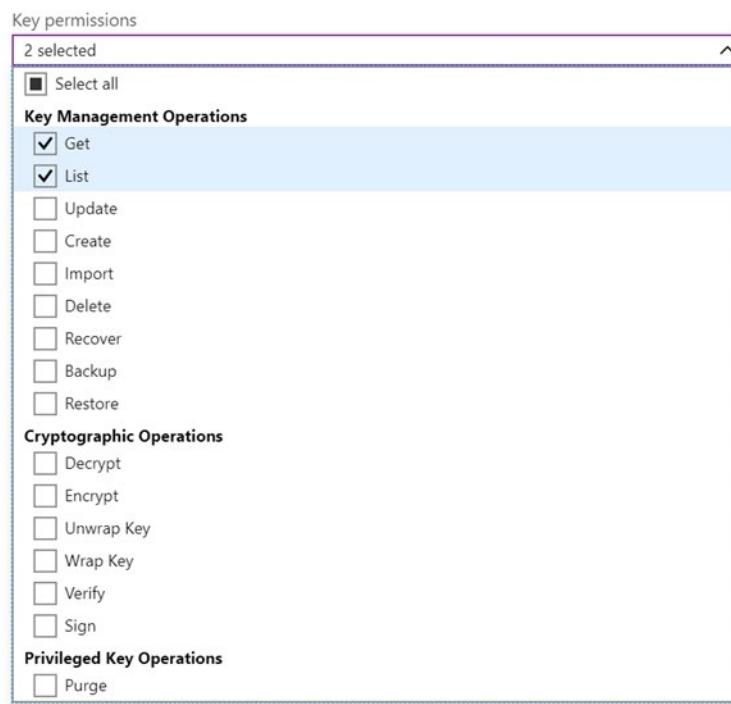
* Microsoft Learning Partner

Authorized application ⓘ

learn-app-dev

Add

The system has a list of predefined management options that define the permissions allowed for this policy - here we have **Key, Secret, & Certificate Management** selected which is appropriate to manage secrets in the Key Vault. You can then customize the permissions as desired by changing the **Key permissions** entries. For example, we could adjust the permissions to only allow *read* operations:



Developers will only need `Get` and `List` permissions to a development-environment vault. A lead or senior developer will need full permissions to the vault to change and add secrets when necessary. Full permissions to production-environment vaults are typically reserved for senior operations staff. For apps, often only `Get` permissions are required as they will just need to retrieve secrets.

Restricting network access

Another point to consider with Azure Key Vault is what services in your network can access the vault. In most cases, the network endpoints don't need to be open to the Internet. You should determine the minimum network access required - for example you can restrict Key Vault endpoints to specific Azure Virtual Network subnets, specific IP addresses, or trusted Microsoft services including Azure SQL, Azure App Service, and various data and storage services that use encryption keys.

The screenshot shows the 'keyvault-xtc - Firewalls and virtual networks' settings page in the Azure portal. The left sidebar has a 'Settings' section with options like Keys, Secrets, Certificates, Access policies, and Firewalls and virtual networks (which is selected and highlighted in blue). The main area shows 'Allow access from:' with 'Selected networks' (radio button) selected (highlighted by a red box). Below this are sections for 'Virtual networks', 'Firewall', and 'IPV4 ADDRESS OR CIDR'.

Exercise - store secrets in Azure Key Vault

Azure sandbox¹

To get some quick experience with Azure Key Vault, let's create a new Key Vault and do the most basic operation available: store a secret. Creating a vault in the Azure portal requires no initial configuration. Your signed-in user identity is automatically granted the full set of secret management permissions, and you can start adding secrets immediately. Once you have a vault, adding and managing secrets can be done from any Azure administrative interface, including the Azure portal, the Azure CLI, and Azure PowerShell.

Create a new Azure Key Vault

Let's start by creating a new Key Vault in the Azure portal.

1. Sign into the [Azure portal²](#) using the same credentials you used to activate the Azure Sandbox.
2. Select **+ Create a resource**.
3. Type **Key Vault** into the search box to find the Azure Key Vault service, and then select **Create**.
4. On the **Basics** tab:
 - Make sure the **Concierge Subscription** is selected in the **Subscription** dropdown.
 - Select [The Sandbox resource group] from the **Resource group** dropdown.
 - Enter a globally unique name for the new vault. Vault names must be 3-24 characters long and contain only alphanumeric characters and dashes. The exercise uses the example name of **VaultamortDiary** for the new vault.
 - Leave the default selected values for **Region** and **Pricing tier**.
5. Select **Review + create** to go to the validation screen.

¹ <https://docs.microsoft.com/learn/modules/configure-and-manage-azure-key-vault/4-store-secrets-in-akv>

² <https://portal.azure.com/>

6. Select **Create** to create the Azure Key Vault.

Once the deployment is complete, navigate to the resource.

Add a secret

Next, add a new secret to the vault.

1. In the Azure portal, select **Secrets** under the **Settings** section of your Azure Key Vault.
2. Click the **Generate/Import** button at the top of the **Secrets** panel.
3. Fill out the **Create a secret** screen with a name, value, and (optional) content type. An example is shown below.

The screenshot shows the 'Create a secret' form in the Azure portal. The URL in the browser is: Home > VaultamortDiary - Overview > VaultamortDiary - Secrets > Create a secret. The form has a title 'Create a secret'. Under 'Upload options', 'Manual' is selected. The 'Name' field is filled with 'HiddenLocation' and has a green checkmark. The 'Value' field contains '.....' and also has a green checkmark. Under 'Content type (optional)', 'Text' is selected. There are two optional checkboxes: 'Set activation date?' and 'Set expiration date?', both with empty input fields. At the bottom, there is a toggle switch labeled 'Enabled?' with 'Yes' selected.

4. Select **Create** to add the secret.

Show the secret

Finally, verify that the secret value has been set.

1. Select your secret from the list and then select the current version of the secret.
2. Select **Show Secret Value** to see the value assigned to the secret.

Other ways to consume the secret

You can create and retrieve secrets from the Azure Key Vault as long as you are authenticated with Azure AD using the REST API, native SDKs, Azure CLI, or Azure PowerShell. For example, here's the same process using Azure PowerShell.

```
Get-AzKeyVault
```

This will return the created vault with the name **VaultamortDiary**.

```
Vault Name      : VaultamortDiary
Resource Group Name : Learn-4f01665a-1272-46a8-9c16-83bbf146494e
Region          : northcentralus
Resource ID     : /subscriptions/xyz/providers/Microsoft.KeyVault/
vaults/VaultamortDiary
```

With the name of the vault and the key, you can retrieve the secret value:

```
Get-AzKeyVaultSecret -VaultName 'VaultamortDiary' -Name 'HiddenLocation'
```

This returns our set value:

```
Vault Name      : vaultamortdiary
Name           : VaultamortDiary
```

```
Version      : ff4b23af35bf4ba9a5c8792227d00ff6
Id          : https://vaultamortdiary1972.vault.azure.net:44
               3/secrets/VaultamortDiary/ff4b23af35bf4ba9
               a5c8792227d00ff6
Enabled     : True
Expires     :
Not Before  :
Created     : 12/17/2020 7:54:03 PM
Updated     : 12/17/2020 7:54:03 PM
Content Type: text
Tags        :
```

NOTE: The module **Manage secrets in your server apps with Azure Key Vault³** shows how to use the Azure CLI and various programming languages to create Key Vaults, set, and retrieve secrets.

Manage certificates

Securely managing certificates is a challenge for every organization. You must ensure that the private key is kept safe, and, as Steve from PetDash found out, certificates have an expiration date and have to be renewed periodically to ensure your website traffic is secure.

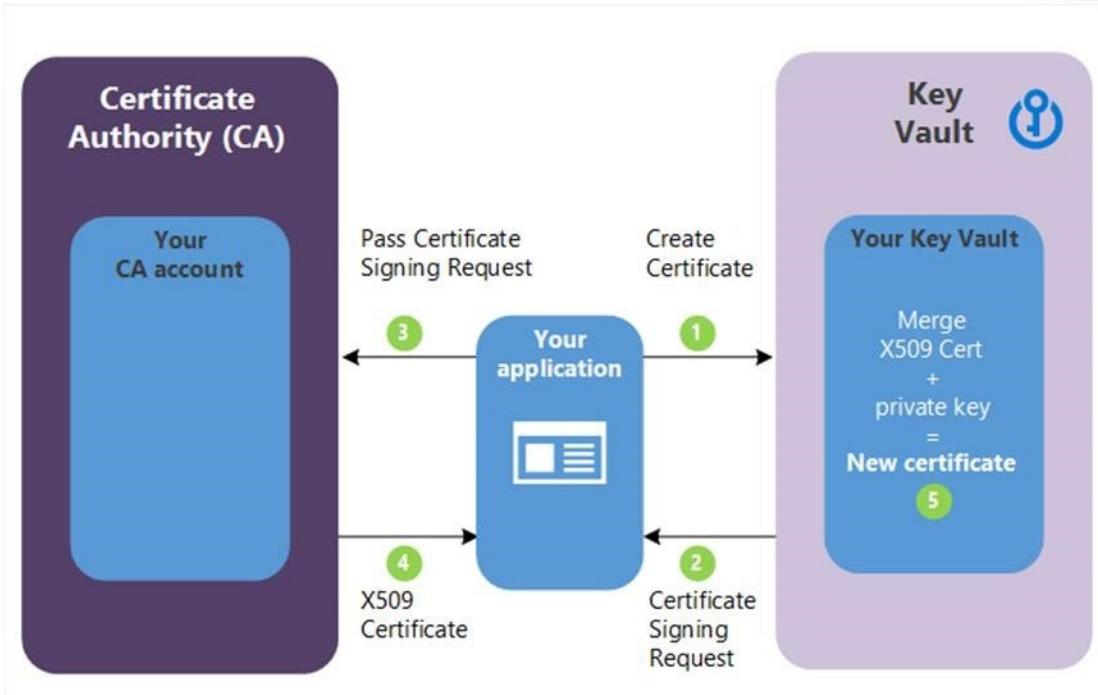
Adding certificates to a Key Vault

Azure Key Vault manages X.509 based certificates that can come from several sources.

First, you can create self-signed certificates directly in the Azure portal. This process creates a public/private key pair and signs the certificate with its own key. These certificates can be used for testing and development.

Second, you can create an X.509 certificate signing request (CSR). This creates a public/private key pair in Key Vault along with a CSR you can pass over to your certification authority (CA). The signed X.509 certificate can then be merged with the held key pair to finalize the certificate in Key Vault as shown in the following diagram.

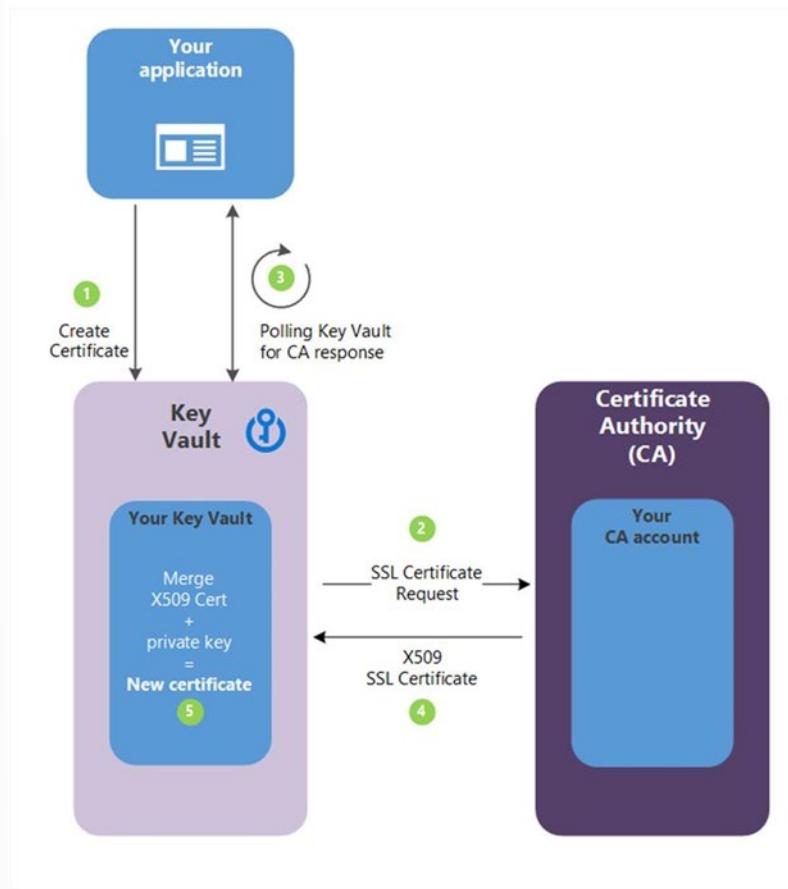
³ <https://docs.microsoft.com/learn/modules/manage-secrets-with-azure-key-vault/>



1. In the previous diagram, your application is creating a certificate which internally begins by creating a key in your Azure Key Vault.
2. Key Vault returns a Certificate Signing Request (CSR) to your application.
3. Your application passes the CSR to your chosen CA.
4. Your chosen CA responds with an X.509 Certificate.
5. Your application completes the new certificate creation with a merger of the X.509 Certificate from your CA.

This approach works with any certificate issuer and provides better security than handling the CSR directly because the private key is created and secured in Azure Key Vault and never revealed.

Third, you can connect your Key Vault with a trusted certificate issuer (referred to as an *integrated CA*) and create the certificate directly in Azure Key Vault. This approach requires a one-time setup to connect the certificate authority. You can then request to create a certificate and the Key Vault will interact directly with the CA to fulfill the request in a similar process to the manual CSR creation process shown above. The full details of this process are presented in the following diagram.



1. In the previous diagram, your application is creating a certificate which internally begins by creating a key in your key vault.
2. Key Vault sends a SSL Certificate Request to the CA.
3. Your application polls, in a loop and wait process, for your Key Vault for certificate completion. The certificate creation is complete when Key Vault receives the CA's response with x509 certificate.
4. The CA responds to Key Vault's SSL Certificate Request with an X509 SSL Certificate.
5. Your new certificate creation completes with the merger of the X509 Certificate for the CA.

This approach has several distinct advantages. Because the Key Vault is connected to the issuing CA, it can manage and monitor the lifecycle of the certificate. That means it can automatically renew the certificate, notify you about expiration, and monitor events such as whether the certificate has been revoked.

Finally, you can import existing certificates - this allows you to add certificates to Key Vault that you are already using. The imported certificate can be in either PFX or PEM format and must contain the private key. For example, here's a PowerShell script to upload a certificate:

```
$pfxFilePath = "C:\WebsitePrivateCertificate.pfx"
$pwd = "password-goes-here"
$flag = [System.Security.Cryptography.X509Certificates.X509KeyStorageFlags]::Exportable
$pkcs12ContentType = [System.Security.Cryptography.X509Certificates.X509ContentType]::Pkcs12
```

```
$collection = New-Object System.Security.Cryptography.X509Certificates.  
X509Certificate2Collection  
$collection.Import($pfxFilePath, $pwd, $flag)  
  
$clearBytes = $collection.Export($pkcs12ContentType)  
$fileContentEncoded = [System.Convert]::ToBase64String($clearBytes)  
$secret = ConvertTo-SecureString -String $fileContentEncoded -AsPlainText  
-Force  
$secretContentType = 'application/x-pkcs12'  
  
# Replace the following <vault-name> and <key-name>.   
Set-AzKeyVaultSecret -VaultName <vault-name> -Name <key-name> -SecretValue  
$secret -ContentType $secretContentType
```

Retrieving certificates from a Key Vault

Once a certificate is stored in your Azure Key Vault, you can use the Azure portal to explore the certificate properties as well as enable or disable a certificate to make it unavailable to clients.

The screenshot shows the Azure Key Vault interface for managing certificates. The top navigation bar includes 'Home', 'Vaultamort - Overview', 'Vaultamort - Certificates', 'petdash-website', and a long ID. Below this, the certificate name '85de4fc420294f02a752ff57b7a6a63e' is displayed with a 'Certificate Version' icon. Action buttons include 'Save', 'Discard', 'Download in CER format', and 'Download in PFX/PEM format'. A 'Properties' section shows 'Created' at 9/4/2019, 3:55:43 PM and 'Updated' at the same time. Under 'Certificate Identifier', the URL 'https://vaultamort.vault.azure.net/certificates/petdash-website/85de4fc420294f02a752ff57b7a6a63e' is shown with a copy icon. The 'Settings' section contains fields for 'Activation Date' (set to 09/04/2019 at 3:45:43 PM, UTC-05:00) and 'Expiration Date' (set to 09/04/2020 at 3:55:43 PM, UTC-05:00). An 'Enabled?' switch is set to 'Yes'. The 'Tags' section shows '0 tags' with a link. The 'Certificate' section displays the subject 'CN=petdash.com' and the issuer 'CN=petdash.com', both with copy icons.

Azure App Service integration

Once you have a public/private key pair certificate in your Azure Key Vault, you can easily associate it to your web app through the Azure portal.

1. Select **TLS/SSL settings** under **Settings**.
2. Select the **Private Key Certificate (.pfx)** tab.
3. Select + **Import Key Vault Certificate** as shown in the following screenshot.

petdash - TLS/SSL settings

Bindings Private Key Certificates (.pfx) Public Key Certificates (.cer)

Private Key Certificate

Private key certificates (.pfx) can be used for TLS/SSL bindings and can be loaded to the certificate store for your app to consume. To understand how to load the certificates for your app to consume click on the learn more link. Uploaded certificates are not available for manual download from the Azure Management Portal, they can only be used by your app hosted on App Service after the required App Settings are set properly or used for TLS/SSL.

[Learn more](#)

Import App Service Certificate **Upload Certificate** **Import Key Vault Certificate** (highlighted with a red box)

Private Key Certificates				
Status Filter	All	Healthy	Warning	Expired
HEALTH STATUS	Hostname	Expiration	Thumbprint	
Healthy	petdash.com	9/4/2020	48755828FB1EE754BDD672E37EA5494BC4B174EA ...	

4. You can then select the vault, which must be in the same subscription, and the secret containing the certificate.
 - The certificate must be an X.509 cert with a content type of application/x-pkcs12 and cannot have a password.

Finally, once the certificate is in place, you'll want to set up a *custom domain*. There's already a built-in certificate for *.azurewebsites.net. You can then associate your custom domain with the certificate you've assigned so the server uses your certificate to secure the connection to the browser.

Summary

Once you have a key vault, you can start using it to store keys and secrets. Your applications no longer need to persist this confidential data, but can request them from the vault as needed. A key vault allows you to update keys and secrets without affecting the behavior of your application, which opens up a breadth of possibilities for your key, secret, and certificate management.

In this module, you've learned about several security benefits of AKV:

- You can create a segmentation of security roles – no one person has the keys to the kingdom.
- The service is monitored and access is logged – this gives you the capability to track user activity to prevent, detect, and minimize a security incident.
- You can avoid human mistakes – other than the creation of the vault, the services can be automated.
- AKV cloud service are available, accessible, and distributed to ensure high reliability for your services.

Further reading

Continue learning about Azure Key Vault in the Microsoft Learn module **Manage secrets in your server apps with Azure Key Vault**⁴.

Read Azure Key Vault documentation on topics we covered in this module.

- **What is Azure Key Vault?**⁵

⁴ <https://docs.microsoft.com/learn/modules/manage-secrets-with-azure-key-vault/>

⁵ <https://docs.microsoft.com/azure/key-vault/key-vault-overview>

- **Azure Key Vault pricing⁶**
- **Certificate operations⁷**
- **Implementing bring your own key (BYOK) for Azure Key Vault⁸**

⁶ <https://azure.microsoft.com/pricing/details/key-vault/>

⁷ <https://docs.microsoft.com/rest/api/keyvault/certificate-operations>

⁸ <https://docs.microsoft.com/azure/key-vault/key-vault-hsm-protected-keys>

Implement compliance controls for sensitive data

Introduction

This module explores the practices of setting compliance controls on the data that is stored within the SQL Server database. We will also review the Azure Defender Advanced Threat Protection options within the Azure environment.

Learning objectives

After taking this module, you will understand:

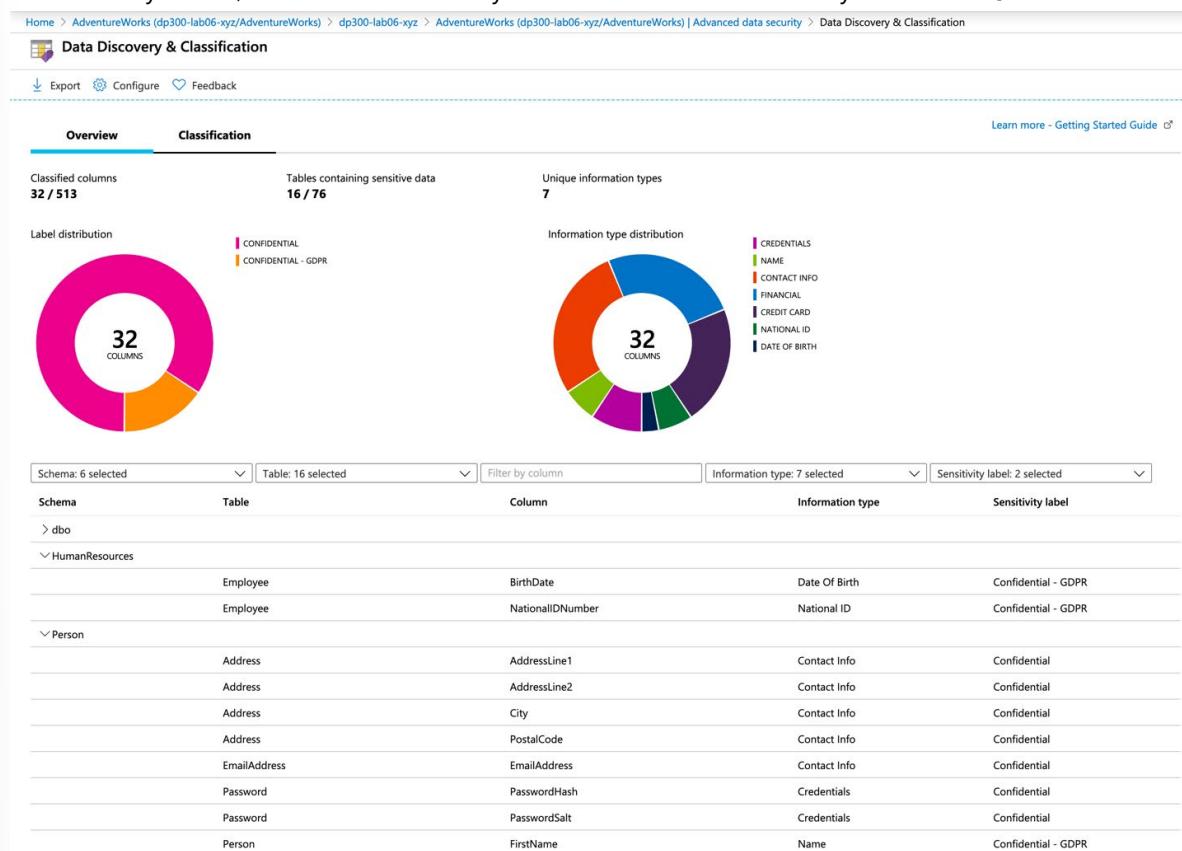
- How data should be classified
- Why data classification should be done
- What Azure threat protection does
- Why Azure threat protection should be used

Describe data classification

Confidential data stored within a Microsoft SQL Server or Azure SQL Database should be classified within the database. This classification allows the SQL Server users as well as other applications to know the sensitivity of the data that is being stored. Data classification with the database is done on a column by column basis. It is possible for a single table to have some columns be public, some columns be confidential, and some columns be highly confidential. Data classification was first introduced into SQL Server Management Studio and used extended properties of objects to store its data classification information. Starting with SQL Server 2019 (and in Azure SQL Database) this metadata is now stored in a catalog view called sys.sensitivity_classifications.

The Azure portal provides a management pane for data classification of your Azure SQL Database as shown below. You can reach this screen by clicking Data Discovery and Classification in the Advanced

Data Security screen, which is in the Security section of the main blade for your Azure SQL Database.



In both the Azure portal and SQL Server Management Studio, you can configure data classification. The classification engine scans your database and locates columns with names that indicate that the column could have sensitive information. One example is that a column named email would be classified by default as containing sensitive personal information. Verify and validate this data, since it is based on column name, so a column named column1 that contained email addresses would not be classified as sensitive personal information.

Columns can be classified using the sensitivity wizard in SQL Server Management Studio, from the Advanced Data Security screen in the Azure portal for Azure SQL Database, or by using the ADD SENSITIVITY CLASSIFICATION T-SQL command as shown below:

```
ADD SENSITIVITY CLASSIFICATION TO
[Application].[People].[EmailAddress]
WITH (LABEL='PII', INFORMATION_TYPE='Email')
GO
```

Classification of data allows you to easily identify the sensitivity of data within the database. Knowing what columns contain sensitive data allows for easier audits and allows you to more easily identify which columns are good choices for data encryption. Classification will allow other employees within the company to make better decisions on how to handle the data which is available within the database.

Describe advanced threat protection

Microsoft offers a suite of protections for Azure SQL Database databases as part of the Advanced Threat Protection (ATP) feature. ATP monitors the connections to the Azure SQL Database databases and the queries which are executed against the database. You can configure ATP from the Azure portal. This screen is reached from the Security section of your databases main blade.

Server settings		Advanced Threat Protecti...								
dp300-lab06-xyz		Learn more - Advanced Threat Protection alerts								
<input type="button" value="Save"/> <input type="button" value="Discard"/> <input type="button" value="Feedback"/>		<input checked="" type="checkbox"/> All <input checked="" type="checkbox"/> SQL injection ⓘ <input checked="" type="checkbox"/> SQL injection vulnerability ⓘ <input checked="" type="checkbox"/> Data exfiltration ⓘ <input checked="" type="checkbox"/> Unsafe action ⓘ <input checked="" type="checkbox"/> Brute Force ⓘ <input checked="" type="checkbox"/> Anomalous client login ⓘ								
ADVANCED DATA SECURITY <div style="display: flex; justify-content: space-around;"> ON OFF </div> <div style="border: 1px solid #ccc; padding: 5px; margin-top: 10px;"> i Advanced Data Security costs [redacted] USD/server/month. It includes Data Discovery & Classification, Vulnerability Assessment and Advanced Threat Protection. We invite you to a trial period for the first 30 days, without charge. </div>										
VULNERABILITY ASSESSMENT SETTINGS <table border="1" style="width: 100%; border-collapse: collapse;"> <tr> <td style="padding: 5px;">Subscription</td> <td style="text-align: right; padding: 5px;">></td> </tr> <tr> <td style="padding: 5px;">Contoso Ltd</td> <td style="text-align: right; padding: 5px;">></td> </tr> <tr> <td style="padding: 5px;">Storage account</td> <td style="text-align: right; padding: 5px;">></td> </tr> <tr> <td style="padding: 5px;">sqlfdbdiag198</td> <td style="text-align: right; padding: 5px;">></td> </tr> </table>			Subscription	>	Contoso Ltd	>	Storage account	>	sqlfdbdiag198	>
Subscription	>									
Contoso Ltd	>									
Storage account	>									
sqlfdbdiag198	>									
Periodic recurring scans ⓘ <div style="display: flex; justify-content: space-around; margin-top: 5px;"> ON OFF </div>										
Send scan reports to ⓘ <div style="display: flex; align-items: center; margin-top: 5px;"> <input type="text" value="demo@contoso.com"/> ✓ </div>										
<input type="checkbox"/> Also send email notification to admins and subscription owners ⓘ										
ADVANCED THREAT PROTECTION SETTINGS										
Send alerts to ⓘ <div style="display: flex; align-items: center; margin-top: 5px;"> <input type="text" value="demo@contoso.com"/> </div>										
<input checked="" type="checkbox"/> Also send email notification to admins and subscription owners ⓘ										
Advanced Threat Protection types <div style="display: flex; justify-content: space-between; align-items: center; margin-top: 5px;"> All > </div>										

To get maximum benefit out of Advanced Threat Protection you will want to enable auditing on your databases. Auditing will allow for deeper investigation into the source of the problem if ATP detects an anomaly. ATP supports alerts for the following threats:

Vulnerability to SQL injection—This alert looks for T-SQL code coming into your database that may be vulnerable to SQL injection attacks. An example would be a stored procedure call that did not sanitize user inputs.

Potential SQL injection—This alert is triggered when an attacker is actively attempting to execute a SQL injection attack.

Access from unusual location—This alert is triggered when a user logs in from an unusual geographic location.

Access from unusual Azure data center—This alert is looking for attacks from an Azure data center that is not normally accessed.

Access from unfamiliar principal—This alert is raised when a user or applications log in to a database that they have not previously accessed.

Access from a potentially harmful application—This alert detects common tools that are used to attack databases.

Brute force SQL credentials—This alert is triggered when there are a high number of login failures with different credentials.

Describe SQL injection

SQL injection is one of the most common methods used for data breaches. The core of the attack is that an SQL command is appended to the back end of a form field in the web or application front end (usually through a website), with the intent of breaking the original SQL Script and then executing the SQL script that was injected into the form field. This SQL injection most often happens when you have dynamically generated SQL within your client application. The core reason behind an SQL Injection attack comes down to poor coding practices both within the client application and within the database stored procedures. Many developers have learned better development practices, but SQL injection is still a significant problem due to both the number of legacy applications still being used and newer applications built by developers who didn't take SQL injection seriously while building the application.

As an example, assume that the front-end web application creates a dynamic SQL statement as shown below:

```
SELECT * FROM Orders WHERE OrderId=25
```

This T-SQL is created when the user goes to the sales order history portion of the company's website and enters 25 into the form field for the order ID number. However, suppose the user enters more than just an ID number, for example "25; DELETE FROM Orders;"

In that case, the query sent to your database would be as shown below:

```
SELECT * FROM Orders WHERE OrderID=25; DELETE FROM Orders;
```

The way the query in the above example works is that the SQL database is told via the semicolon ";" that the statement has ended and that there is another statement that should be run. The database then processes the next statement as instructed, which would result in the deletion of all rows from the Orders table.

The initial SELECT query is run as normal without any errors being generated. However, when you look at the Orders table, you won't see any rows. The second query in the batch, which deletes all the rows, was also executed.

One technique used to prevent SQL injection attacks is to inspect the text of the parameters, or values entered into the form fields, looking for various keywords. However, this solution only provides minimum protection as there are many, many ways to force these attacks to work. Some of these injection techniques include passing in binary data, having the database engine convert the binary data back to a text string, and then executing the string. You can see a simple example of this problem by running the T-SQL code below.

```
DECLARE @v varchar(255)  
  
SELECT @v = cast(0x73705F68656C706462 as varchar(255))
```

```
EXEC  (@v)
```

When data is being accepted from a user, either a customer or an employee, one good way to ensure that the value won't be used for an SQL injection attack is to validate that the data that was entered is of the expected data type. If a number is expected, the client application should ensure that there is in fact a number being returned. If a text string is expected, then ensure that the text string is of the correct length, and it does not contain any binary data within it. The client application should be able to validate all data being passed in from the user. Validation can be done either by informing the user of the problem and allowing the user to correct the issue, or by crashing gracefully in such a way that an error is returned and no commands are sent to the database or the file system.

While fixing your application code should always be the priority, in some cases that may not be possible, so having Advanced Threat Protection can provide an additional layer of protection for your sensitive data.

Exercise- Enable Advanced Data Security and Data Classification

You've been hired as a Senior Database Administrator help ensure the security of the database environment. These tasks will focus on Azure SQL Database. You will learn how to configure advanced security and classification.

Enable Advanced Data Security and Data Classification

1. When the VM lab environment opens use the password on the **Resources** tab for the Student account to sign in to Windows.
2. Select the Microsoft Edge browser from the toolbar and navigate to <https://portal.azure.com>⁹. This should be the home page of the browser.
3. The username and password information to sign into the Azure portal are in the **Resources** tab above these instructions. If you select the **Username**, it will be filled in for you.
4. Select **Next**.
5. Select the **Password** field to have that filled in for you, then select **Sign in**.
6. Select **Yes**.
7. On the **Welcome to Azure** popup, select **Maybe later**.
8. On the Azure portal home page, select **All resources**.

⁹ <https://portal.azure.com/>

The screenshot shows the Microsoft Azure portal interface. At the top, there's a navigation bar with links for Home - Microsoft Azure, https://portal.azure.com/#home, and various account settings. Below the bar, the main content area is titled "Azure services". It features several icons for different services: Create a resource (plus sign), All resources (grid), Virtual machines, App Services, Storage accounts, SQL databases, Azure Database for PostgreSQL, and Azure Cosmos DB. Below these are two more sections: "Kubernetes services" and "More services" (with an arrow icon). Under "Navigate", there are links for Subscriptions, Resource groups, and All resources (which is highlighted with a red box). Further down is a "Dashboard" link. In the "Tools" section, there are links for Microsoft Learn, Azure Monitor, and Security Center.

9. Select the Azure SQL Database server starting with **azuresql-lab**.

This screenshot shows the "All resources" blade in the Azure portal. The title bar says "All resources" and includes a "Home >" link and a close button. Below the title are filter options: "Filter by name...", "Subscription == all", "Resource group == all", "Type == all", "Location == all", and "Add filter". There are also buttons for "Assign tags", "Delete", and "Feedback". The main table displays two records:

Name	Type	Resource group	Location	Subscription
AdventureWorksLT (azuresql-lab-x77lt2vnf7fbu...)	SQL database	secureSQLLab	East US	MOC Subscription-1d74...
azuresql-lab-x77lt2vnf7fbu	SQL server	secureSQLLab	East US	MOC Subscription-1d74...

10. From the main blade of your Azure SQL server, navigate to the **Security** section, and select **Security Center**.

The screenshot shows the Azure portal interface for managing a SQL server. The top navigation bar includes 'Home > All resources > azuresql-lab-x77lt2vnf7fbu'. The left sidebar lists several categories: Active Directory admin, SQL databases, SQL elastic pools, Deleted databases, Import/Export history, DTU quota, Properties, Locks, Security (Auditing, Firewalls and virtual networks, Private endpoint connections), and a highlighted 'Security Center' link. The main content area displays the server's configuration under the 'Essentials' tab, including resource group (secureSQLLab), status (Available), location (East US), subscription (MOC Subscription-ld7421487), and server name (azuresql-lab-x77lt2vnf7fbu.database.windows.net). A 'Tags (change)' section shows 'displayName : SqlServer'. Below this, there are tabs for 'Notifications (0)' and 'Features (6)', with 'Features (6)' selected. Under 'Features (6)', a card for 'Active Directory admin' is shown, stating it allows central management of identity and access to Azure SQL databases. At the bottom of the main content area, there are buttons for 'All', 'Security (4)', 'Performance (1)', and 'Recovery (1)'.

11. Select the toggle switch under **AZURE DEFENDER FOR SQL** to **ON**, and then select **Storage account**.

The screenshot shows the 'azuresql-lab-7eqq3aokgguhq' Security Center page. The left sidebar includes sections for Properties, Locks, Security (Auditing, Firewalls and virtual networks, Private endpoint connections), a highlighted 'Security Center' link, and Transparent data encryption. The main content area features the 'AZURE DEFENDER FOR SQL' section, which has a 'ON' toggle switch (highlighted with a red box) and an information card about the service cost and trial period. Below this is the 'VULNERABILITY ASSESSMENT SETTINGS' section, which includes a 'Subscription' dropdown set to 'MOC Subscription-ld7421488' and a 'Storage account' dropdown (highlighted with a red box). Further down are sections for 'Periodic recurring scans' (ON/OFF switch) and 'Send scan reports to' (with an info icon).

On the **Choose storage account** page, select + **Create new**.

12. On the **Create storage account** page, in the **Name** field enter a unique name starting with **securesqlab** and append some additional random numbers until you have a valid name.

Home > azuresql-lab-7eqq3aokgguhq > Choose storage account >

Create storage account

Name *

 ✓
.core.windows.net

Account kind ⓘ

Storage (general purpose v1)

Performance ⓘ

Standard Premium

Replication ⓘ

Locally-redundant storage (LRS)

OK

13. Select **OK** and wait for the storage account to be created.
14. Toggle the switch for **Periodic recurring scans** to **ON**. Enter the Azure account email in the **Send scan reports to** and **Send alerts to** dialog boxes. Deselect **Also send email notification to admins and subscription owners**.
- Select **Advanced Threat Detection types** and review the selections. Leave all of the boxes checked and select **OK**.

The screenshot shows the Azure Security Center settings page for a specific Azure SQL server. The left navigation menu includes options like Properties, Locks, Security (selected), Auditing, Firewalls and virtual networks, Private endpoint connections, Security Center (selected), Transparent data encryption, Intelligent Performance (Automatic tuning, Recommendations), Monitoring (Logs), and Automation. The main content area shows a storage account named 'securesqllab12345'. Under 'Periodic recurring scans', the 'ON' button is selected. A note states: 'Scans will be triggered automatically once a week. In most cases, it will be on the day Vulnerability Assessment has been enabled and saved. A scan result summary will be sent to the email addresses you provide.' Below this, there's a section to 'Send scan reports to' with an input field containing 'User1-14242301@lodazurepass.onmicrosoft.com' (highlighted with a red box). There's also an unchecked checkbox for 'Also send email notification to admins and subscription owners'. The 'ADVANCED THREAT PROTECTION SETTINGS' section includes a 'Send alerts to' input field with the same email address (highlighted with a red box) and an unchecked checkbox for 'Also send email notification to admins and subscription owners'. A note below says 'Advanced Threat Protection types: All' with a 'View details' link (highlighted with a red box). At the bottom, there's a link to 'Enable Auditing for better threats investigation experience'.

15. Select **Save**

Enable data classification on an Azure SQL database

1. In the left navigation, select **Overview**.
2. Copy the **Server name** to use later in this exercise.

Home > All resources > azuresql-lab-jcm4xkk65uuc2 > AdventureWorksLT (azuresql-lab-jcm4xkk65uuc2/AdventureWorksLT) >

azuresql-lab-jcm4xkk65uuc2

SQL server

Search (Ctrl+ /)

Overview

Create database New elastic pool New Synapse SQL pool (data warehouse) Import database ...

Resource group (change)
secureSQLLab

Status Available

Location East US

Subscription (change)
MOC Subscription-10829867

Subscription ID
8d91bf2d-955f-4a25-a116-1b85fc0d01cb

Tags (change)
displayName : SqlServer

Server admin labadmin

Firewalls and virtual networks Show firewall settings

Active Directory admin Not configured

Server name azuresql-lab-jcm4xkk65uuc2.database.windows.net

Copy to clipboard

Notifications (1) Features (6)

All Info (1) Recommendations (0)

Azure Defender for SQL Free Trial
Your free trial will expire in 30 days

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

Quick start Failover groups Manage Backups Active Directory admin SQL databases SQL elastic pools Deleted databases Import/Export history

3. Navigate to the **AdventureWorksLT** database in the Azure portal by scrolling down in the overview screen for Azure SQL server and select the database name.

Home > All resources > azuresql-lab-jcm4xkk65uuc2 > AdventureWorksLT (azuresql-lab-jcm4xkk65uuc2/AdventureWorksLT) >

azuresql-lab-jcm4xkk65uuc2

SQL server

Search (Ctrl+ /)

Create database New elastic pool New Synapse SQL pool (data warehouse) Import database ...

Azure Defender for SQL Free Trial
Your free trial will expire in 30 days

Available resources

Filter by name All types

Name	Type	Status	Pricing tier
AdventureWorksLT	SQL database	Online	General Purpose: Gen5, 2 ...

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

Quick start Failover groups Manage Backups Active Directory admin SQL databases SQL elastic pools Deleted databases Import/Export history

4. Navigate to the Security section of the main blade for your Azure SQL Database and select **Data Discovery & Classification**.

The screenshot shows the Azure portal interface for the 'AdventureWorksLT' database. The left sidebar lists various database management options. The 'Data Discovery & Classification' link is highlighted with a red box. The main content area displays the database's essential properties, including its resource group ('secureSQLLab'), server name ('azuresql-lab-jcm4xkk65uuc2.database.windows.net'), and subscription information ('MOC Subscription-10d10829867'). It also shows compute utilization metrics for the last hour, 24 hours, and 7 days.

- On the **Data Discovery & Classification** screen you will see an informational message that reads **We have found 15 columns with classification recommendations**. Select that link.

The screenshot shows the 'Data Discovery & Classification' screen for the 'AdventureWorksLT' database. The left sidebar shows the 'Data Discovery & Classification' link selected. A prominent message at the top states, 'We have found 15 columns with classification recommendations →', which is also highlighted with a red box. The main area displays two tabs: 'Overview' (selected) and 'Classification'. Below these are several data points and visualizations. The 'Overview' tab shows 0 classified columns out of 109, 0 tables containing sensitive data out of 12, and 0 unique information types. The 'Classification' tab shows 0 label distribution and 0 information type distribution. Two donut charts are present, both showing 0 columns.

- On the next **Data Discovery & Classification** screen select the check box next to **Select all**, select **Accepted selected recommendations**, and then select **Save** to save the classifications into the database.

The screenshot shows the Microsoft Azure Data Discovery & Classification interface for the AdventureWorksLT database. The 'Classification' tab is selected. A red box highlights the 'Save' button at the top left. Another red box highlights the 'Accept selected recommendations' button. A third red box highlights the 'Select all' checkbox in the table header. The table lists 15 columns with their schema, table, column name, information type, and sensitivity label. Most columns are labeled 'Customer' or 'SalesLT' and have 'Confidential - GDPR' as the sensitivity label.

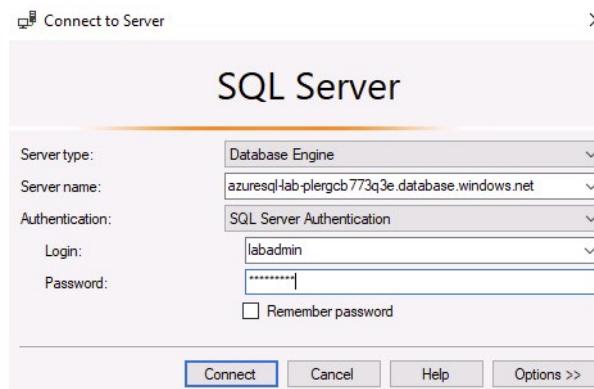
Schema	Table	Column	Information type	Sensitivity label
SalesLT	Customer	FirstName	Name	Confidential - GDPR
SalesLT	Customer	LastName	Name	Confidential - GDPR
SalesLT	Customer	EmailAddress	Contact Info	Confidential
SalesLT	Customer	Phone	Contact Info	Confidential
SalesLT	Customer	PasswordHash	Credentials	Confidential
SalesLT	Customer	PasswordSalt	Credentials	Confidential
dbo	ErrorLog	UserName	Credentials	Confidential
SalesLT	Address	AddressLine1	Contact Info	Confidential
SalesLT	Address	AddressLine2	Contact Info	Confidential
SalesLT	Address	City	Contact Info	Confidential
SalesLT	Address	PostalCode	Contact Info	Confidential
SalesLT	CustomerAddress	AddressType	Contact Info	Confidential
SalesLT	SalesOrderHeader	AccountNumber	Financial	Confidential
SalesLT	SalesOrderHeader	CreditCardApprovalCode	Credit Card	Confidential
SalesLT	SalesOrderHeader	TaxAmt	Financial	Confidential

View data classification in SQL Server Management Studio

NOTE: If you'd like to copy and paste the code you can find the code in the **D:\LabFiles\Secure Environment\exercise_steps.sql** file.

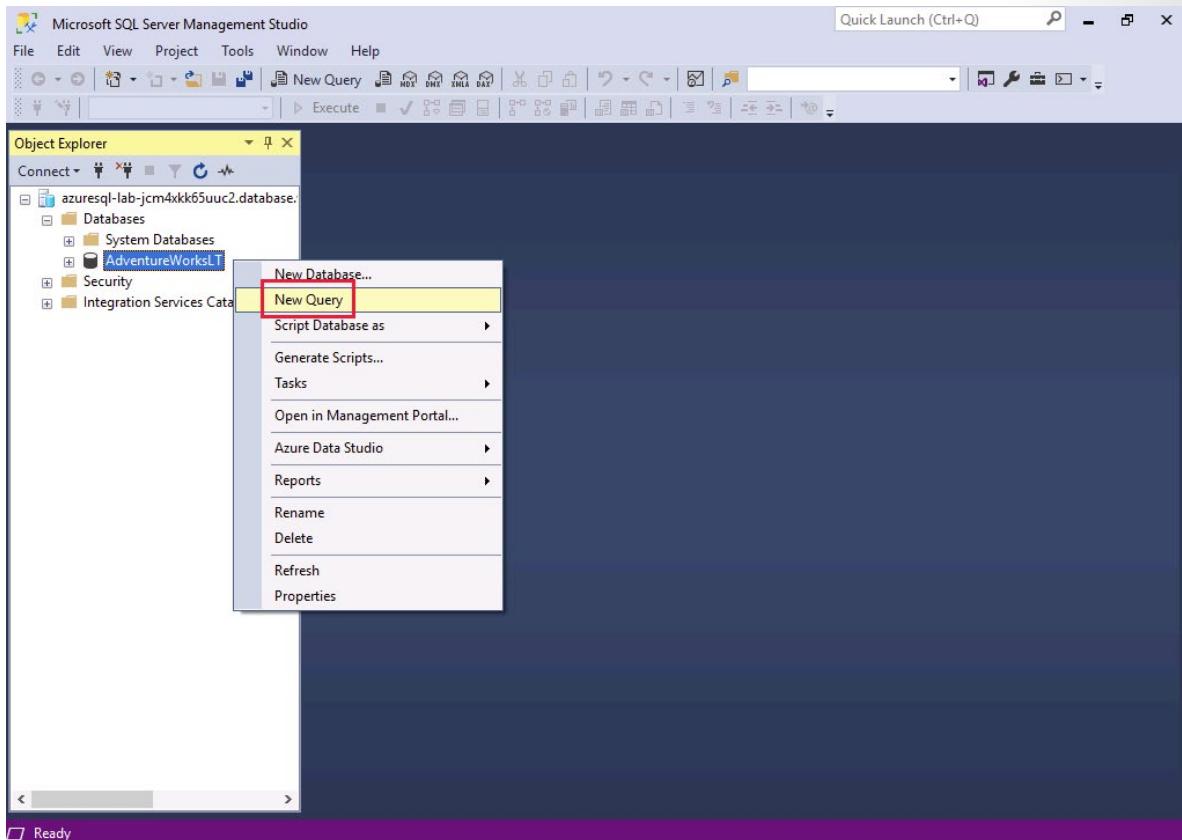
1. Open SQL Server Management Studio by navigating to **Microsoft SQL Server Tools 18 > SQL Server Management Studio** from the Start menu. Paste in the name of your Azure SQL database server and login with these credentials:

- Server admin login: **labadmin**
- Password: **Azur3Pa\$\$**



Select **Connect**.

2. You'll be prompted to add your client IP address as a new firewall rule. Sign in with the Azure credentials in the Resources tab. Then select **OK**.
3. In the **Object Explorer**, expand the server node, and expand the **Databases** node.



4. Right-click on the **AdventureWorksLT** database, and select **New Query**.
5. Execute the following query in the new query window.

```
SELECT o.name AS [Table Name]
,ac.name AS [Column Name]
,sc.label
,sc.information_type
FROM sys.sensitivity_classifications sc
INNER JOIN sys.objects o ON o.object_id = sc.major_id
INNER JOIN sys.all_columns ac ON ac.column_id = sc.minor_id
WHERE ac.object_id = o.object_id;
```

This query will return the results of your classified columns as shown below.

The screenshot shows a SQL Server Management Studio window. The query editor contains the following T-SQL code:

```
SELECT o.name AS [Table Name]
      ,ac.name AS [Column Name]
      ,sc.label
      ,sc.information_type
  FROM sys.sensitivity_classifications sc
 INNER JOIN sys.objects o ON o.object_id = sc.major_id
 INNER JOIN sys.all_columns ac ON ac.column_id = sc.minor_id
 WHERE ac.object_id = o.object_id;
```

The results grid displays 15 rows of data from the sys.sensitivity_classifications catalog view. The columns are Table Name, Column Name, label, and information_type. The data includes various columns from tables like Customer, ErrorLog, and Address, each assigned a specific sensitivity label and type.

	Table Name	Column Name	label	information_type
1	Customer	EmailAddress	Confidential	Contact Info
2	Customer	FirstName	Confidential - GDPR	Name
3	Customer	LastName	Confidential - GDPR	Name
4	Customer	PasswordHash	Confidential	Credentials
5	Customer	PasswordSalt	Confidential	Credentials
6	Customer	Phone	Confidential	Contact Info
7	ErrorLog	UserName	Confidential	Credentials
8	Address	AddressLine1	Confidential	Contact Info
9	Address	AddressLine2	Confidential	Contact Info
10	Address	City	Confidential	Contact Info
11	Address	PostalCode	Confidential	Contact Info

At the bottom of the screen, the status bar shows "Query executed successfully.", the connection details "azuresql-lab-7eqq3aokgguhq...", the user "labadmin (94)", the database "AdventureWorksLT", and the duration "00:00:00". A red box highlights the "15 rows" message.

Note that the results show that the 15 classifications have been created in the database.

In this exercise you've seen how the Azure portal can automatically classify sensitive columns in a database table for you.

To finish this exercise select **Done** below.

Knowledge check

Question 1

Where is the data from data classification stored in SQL Server 2019?

- In the extended properties for each object.
- In the sys.sensitivity_classifications catalog view.
- In the sys.all_columns catalog view

Question 2

Which of the following threats is analyzed by Advanced Threat Protection?

- Weak passwords.
- Open Firewall rules.
- SQL Injection.

Question 3

Which attack type is commonly associated with dynamic SQL?

- SQL Injection.
- Brute Force.
- Data Exfiltration.

Summary

This module explored the practices of setting compliance controls on the data that is stored within the SQL Server database. You also reviewed the Azure Defender Advanced Threat Protection options within the Azure environment.

Now that you've reviewed this module, you should be able to:

- Describe how data should be classified
- Describe why data classification should be done
- Describe what Azure Threat Protection does
- Describe why Azure Threat Protection should be used

Module Lab information

Lab 8 - End-to-end security with Azure Synapse Analytics

- **Estimated Time:** 50 - 60 minutes
- **Lab files:** The files for this lab are located in the *Instructions\Labs\13* folder.

Lab overview

In this lab, students will learn how to secure a Synapse Analytics workspace and its supporting infrastructure. The student will observe the SQL Active Directory Admin, manage IP firewall rules, manage secrets with Azure Key Vault and access those secrets through a Key Vault linked service and pipeline activities. The student will understand how to implement column-level security, row-level security, and dynamic data masking when using dedicated SQL pools.

Lab objectives

After completing this lab, you will be able to:

- Secure Azure Synapse Analytics supporting infrastructure
- Secure the Azure Synapse Analytics workspace and managed services
- Secure Azure Synapse Analytics workspace data

Module summary

module-summary

In this module, you have learned how to approach and implement security to protect your data with Azure Synapse Analytics.

Learning objectives

In this module, you have:

- Secured a data warehouse
- Configured and manage secrets in Azure Key Vault
- Implemented compliance controls for sensitive data

Post Course Review

After the course, consider visiting **Azure security baseline for Synapse Analytics¹⁰**. The Azure Security Benchmark provides recommendations on how you can secure your cloud solutions on Azure.

Important

Remember

Should you be working in your own subscription while running through this content, it is best practice at the end of a project to identify whether or not you still need the resources you created. Resources left running can cost you money.

You can delete resources one by one, or just delete the resource group to get rid of the entire set.

¹⁰ <https://docs.microsoft.com/en-us/azure/synapse-analytics/security-baseline>

Answers

Question 1

You want to configure a private endpoint. You open up Azure Synapse Studio, go to the manage hub, and see that the private endpoints is greyed out. Why is the option not available?

- Azure Synapse Studio does not support the creation of private endpoints.
- A conditional access policy has to be defined first.
- A managed virtual network has not been created.

Explanation

Correct. In order to create a private endpoint, you first must create a managed virtual network.

Question 2

You require you Azure Synapse Analytics Workspace to access an Azure Data Lake Store using the benefits of the security provided by Azure Active Directory. What is the best authentication method to use?

- Storage account keys.
- Shared access signatures.
- Managed identities.

Explanation

Correct. Managed identities provides Azure services with an automatically managed identity in Azure Active Directory. You can use the Managed Identity capability to authenticate to any service that support Azure Active Directory authentication.

Question 1

Where is the data from data classification stored in SQL Server 2019?

- In the extended properties for each object.
- In the sys.sensitivity_classifications catalog view.
- In the sys.all_columns catalog view

Explanation

That's correct. This is the correct view.

Question 2

Which of the following threats is analyzed by Advanced Threat Protection?

- Weak passwords.
- Open Firewall rules.
- SQL Injection.

Explanation

That's correct. One of the key features of ATP is SQL Injection inspection.

Question 3

Which attack type is commonly associated with dynamic SQL?

- SQL Injection.
- Brute Force.
- Data Exfiltration.

Explanation

That's correct. SQL Injection attacks are common with poorly coded dynamic SQL.

Module 9 Support Hybrid Transactional Analytical Processing (HTAP) with Azure Synapse Link

Module introduction

module-introduction

In this module, students will learn how to perform operational analytics against Azure Cosmos DB using the Azure Synapse Link feature within Azure Synapse Analytics.

Learning objectives

In this module, you will:

- Design hybrid transactional and analytical processing using Azure Synapse Analytics
- Configure Azure Synapse Link with Azure Cosmos DB
- Query Azure Cosmos DB with Apache Spark for Azure Synapse Analytics
- Query Azure Cosmos DB with SQL Serverless for Azure Synapse Analytics

Design hybrid transactional and analytical processing using Azure Synapse Analytics

Introduction

Learn how Azure Synapse Analytics Link for Cosmos DB solves the issue of making operational data available for analytical query in near real time.

In this module, you will:

- Understand Hybrid Transactional and Analytical Processing patterns
- Describe the supported analytical workloads
- Explain HTAP scenarios in the context of Synapse Link
- Describe the analytical store
- Understand well-defined schemas
- Design schemas to support type of analytics
- Understand writeback scenarios

Prerequisites

Before taking this module, it is recommended that you are able to:

- Have a good understanding of Azure Cosmos DB capabilities
- Have a working knowledge of SQL or Spark

Understand hybrid transactional and analytical processing patterns

Many business application architectures separate transactional and analytical processing into separate systems with data stored and processed on separate infrastructures. These infrastructures are commonly referred to as OLTP (online transaction processing) systems working with operational data, and OLAP (online analytical processing) systems working with historical data, with each system optimized for their specific task.

OLTP systems are optimized for dealing with discrete system or user requests immediately and responding as quickly as possible.

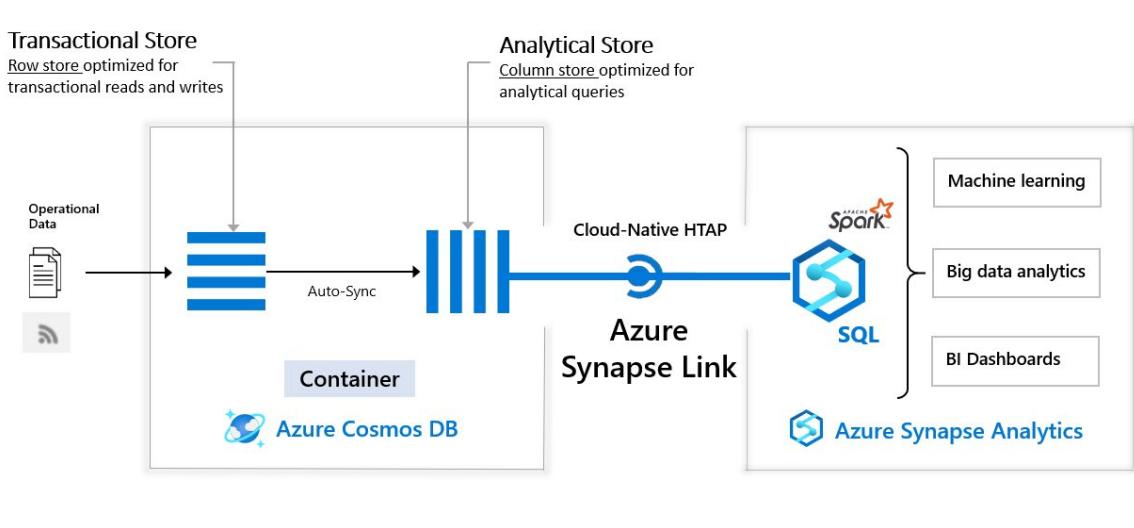
OLAP systems are optimized for the analytical processing, ingesting, synthesizing, and managing large sets of historical data. The data processed by OLAP systems largely originates from OLTP systems and needs to be loaded into the OLTP systems by means of batch processes commonly referred to as ETL (Extract, Transform, and Load) jobs.

Due to their complexity and the need to physically copy large amounts of data, this creates a delay in data being available to provide insights by way of the OLAP systems.

As more and more businesses move to digital processes, they increasingly recognize the value of being able to respond to opportunities by making faster and well-informed decisions. HTAP (Hybrid Transactional/Analytical processing) enables business to run advanced analytics in near-real-time on data stored and processed by OLTP systems.

Azure Synapse Link for Azure Cosmos DB

Azure Synapse Link for Azure Cosmos DB is a cloud-native HTAP capability that enables you to run near-real-time analytics over operational data stored in Azure Cosmos DB. Azure Synapse Link creates a tight seamless integration between Azure Cosmos DB and Azure Synapse Analytics.



Azure Cosmos DB provides both a transactional store optimized for transactional workloads and an analytical store optimized for analytical workloads and a fully managed autosync process to keep the data within these stores in sync.

Azure Synapse Analytics provides both a SQL Serverless query engine for querying the analytical store using familiar T-SQL and an Apache Spark query engine for leveraging the analytical store using your choice of Scala, Java, Python or SQL and provides a user-friendly notebook experience.

Together Azure Cosmos DB and Synapse Analytics enable organizations to generate and consume insights from their operational data in near-real time, using the query and analytics tools of their choice. All of this is achieved without the need for complex ETL pipelines and without affecting the performance of their OLTP systems using Azure Cosmos DB.

Describe the supported analytical workloads

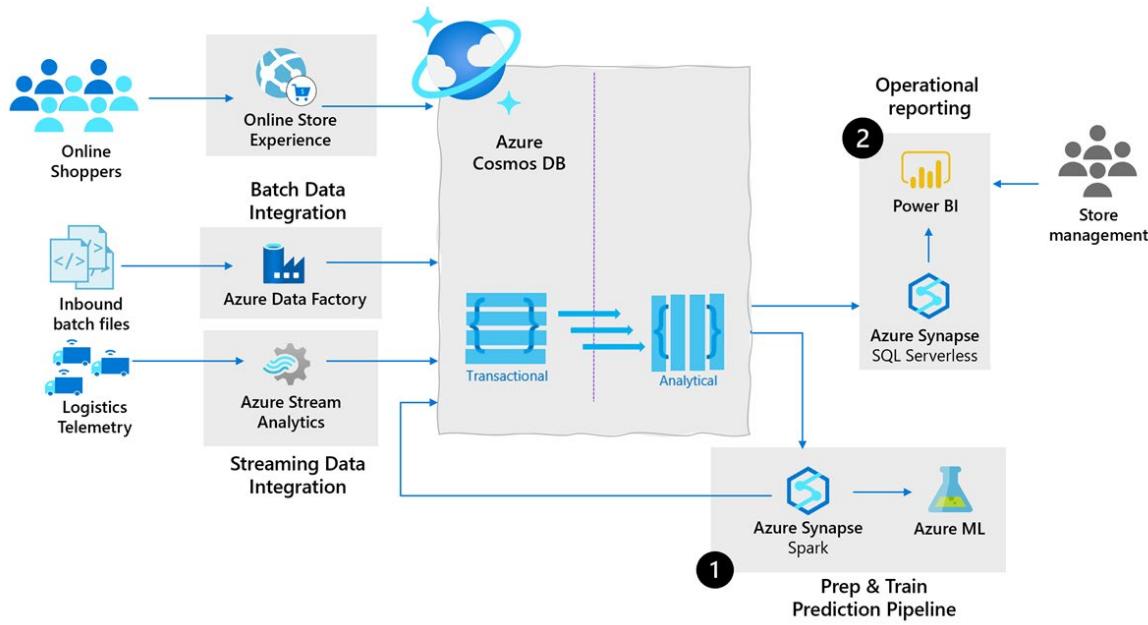
The following are common use cases for using Azure Synapse Link for Azure Cosmos DB capability to address real-world business needs using operational analytics:

Supply chain analytics, forecasting and reporting.

With supply chains generating increasing volumes of operational data every minute for orders, shipments and sales transactions, manufacturers and retailers need an operational database that can scale to handle the data volumes as well as an analytical platform to get to a level of real-time contextual intelligence to stay ahead of the curve.

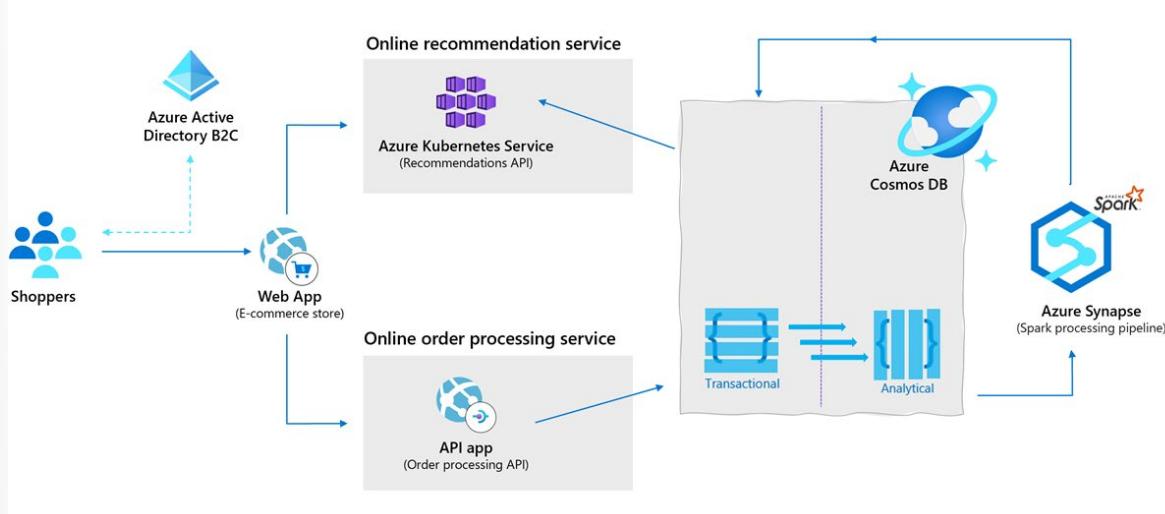
Azure Synapse Link for Cosmos DB allows these organizations to store data from their sales systems, ingest real-time telemetry data from in vehicle systems and integrate data from their ERP systems into a common operational store in Azure Cosmos DB and then leverage the data from Synapse analytics to enable both predictive analytics scenarios such as stock out monitoring and supply chain bottleneck

management (1) in addition to enabling operational reporting directly on their operation data using standard reporting tools such as Power BI (2).



Retail real-time personalization.

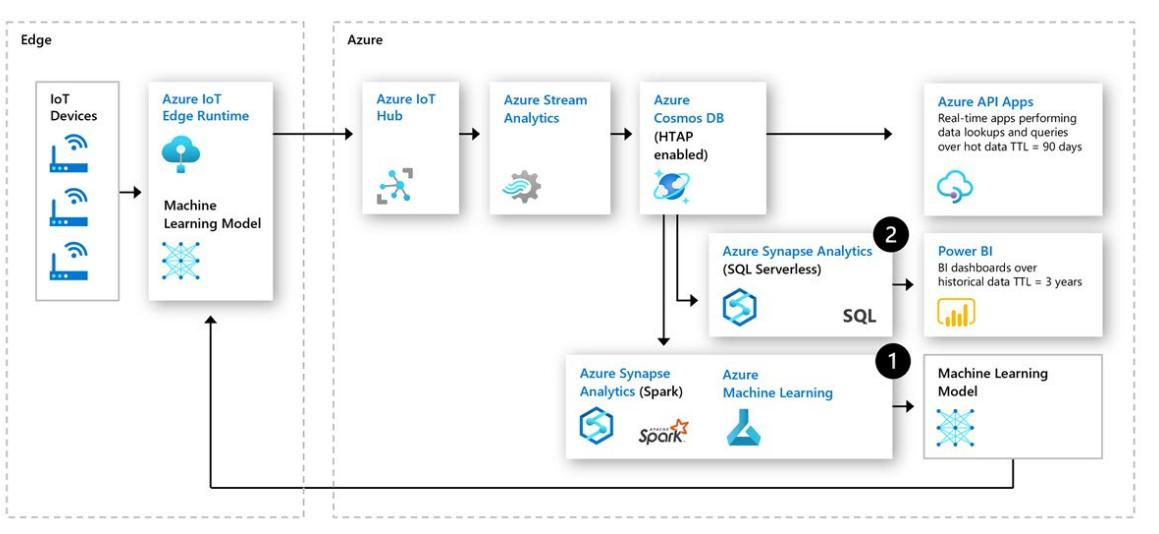
In retail, many web-based retailers will perform real-time basket analysis to make product recommendations to customers who are about to purchase products. This increased revenues for these organizations as the provided targeted suggestions at the point of sales



Predictive maintenance using anomaly detection with IOT

Industrial IoT innovations have drastically reduced downtimes of machinery and increased overall efficiency across all fields of industry. One of such innovations is predictive maintenance analytics for machinery at the edge of the cloud.

The following architecture leverages the cloud native HTAP capabilities of Azure Synapse Link for Azure Cosmos DB in IoT predictive maintenance:



Explain HTAP scenarios in the context of Synapse Link

HTAP enables businesses to perform analytics over database systems that provide high performance transactional capabilities without impacting the performance of these systems. This enables organizations to use a database to fulfill both transactional and analytical needs to support near real-time analysis of operational data to make timely decisions.

As an example, Adventure Works uses Azure Cosmos DB to store sales orders and customer profile data from their eCommerce site. The NoSQL document store provided by the Azure Cosmos DB provides the familiarity of managing their data using SQL syntax, while being able to read and write the files at a massive, global scale.

While Adventure Works is happy with the capabilities and performance of Azure Cosmos DB, they are concerned about the cost of executing a large volume of complex analytical queries needed to fulfill their operational reporting requirements. They want to efficiently access all their operational data stored in Cosmos DB without needing to increase the Azure Cosmos DB throughput and associate cost. They have looked at options for extracting data from their containers to the data lake as it changes, through the Azure Cosmos DB change feed mechanism. The problem with this approach is the extra service and code dependencies and long-term maintenance of the solution. They could perform bulk exports from a Synapse Pipeline, but then they won't have the most up-to-date information at any given moment.

They decide to enable Azure Synapse Link for Cosmos DB and enable the analytical store on their Azure Cosmos DB containers. With this configuration, all transactional data is automatically stored in a fully isolated column store. This store enables large-scale analytics against the operational data in Azure Cosmos DB, without impacting the transactional workloads or incurring additional costs. Azure Synapse Link for Cosmos DB creates a tight integration between Azure Cosmos DB and Azure Synapse Analytics, which enables Adventure Works to run near real-time analytics over their operational data with no-ETL and full performance isolation from their transactional workloads.

By combining the distributed scale of Cosmos DB's transactional processing and the built-in analytical store with the computing power of Azure Synapse Analytics, Azure Synapse Link enables a Hybrid Transactional/Analytical Processing (HTAP) architecture for optimizing Adventure Works business pro-

cesses. This integration eliminates ETL processes, enabling business analysts, data engineers and data scientists to self-serve and run near real-time BI, analytics, and Machine Learning pipelines over operational data.

Describe the analytical store

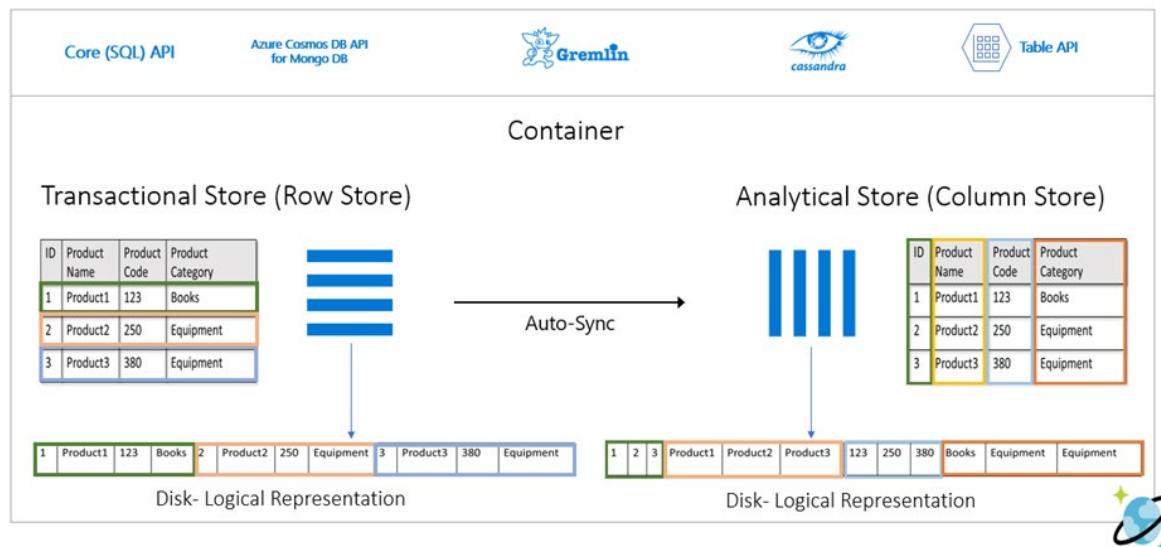
Azure Cosmos DB analytical store is a fully isolated column store for enabling large-scale analytics against operational data in your Azure Cosmos DB, without any impact to your transactional workloads.

Row-oriented transactional store

Operational data in an Azure Cosmos DB container is internally stored in an indexed row-based “transactional store”. The row store format and its associated b-tree index are designed to allow fast transactional reads and writes with single-digit millisecond response times, and high-performance operational queries. As your dataset grows large, complex analytical queries can become expensive as they use up more of the provisioned throughput resources. The increased consumption of provisioned throughput in turn impacts the performance of transactional workloads.

Column-oriented analytical store

Azure Cosmos DB analytical store addresses the complexity and latency challenges that occur with the traditional ETL pipelines. Azure Cosmos DB analytical store can automatically sync data from the transactional store into a separate column store. Column store format is suitable for large-scale analytical queries to be performed in an optimized manner, resulting in improving the latency of such queries.



Features of the analytical store

When you create an Azure Cosmos DB container you have the option of enabling analytical store, a new column-store structure is created within the container duplicating the data of the transactional store. This column store structure data is persisted separately from the row-oriented transactional store with the inserts, updates, and deletes performed on the transactional store being transparently copied by means of a fully managed internal autosync process to the analytical store in near real time.

Note: You can only enable analytical store at the time of creating a new container.

Note: Azure Synapse Link is supported for the Azure Cosmos DB SQL (Core) API and for the Azure Cosmos DB API for MongoDB.

Data is typically automatically synchronized between the transactional store and the analytical store within 2 minutes by means of the autosync process. However, in some circumstances -most notably in situations shared throughput database with many containers, the autosync latency could take up to 5 minutes.

Due to the fact that the transactional store and analytical store are persisted and queried separately the workloads associated with these stores are isolated from each other, that is to say queries against the analytical store (or the autosync process itself) does not impact the performance of nor use up resources (throughput or request units) provisioned for the transactional store, and operations performed against the transactional store does not impact autosync latency.

Note: The transactions (read & write) and storage costs for the analytical store are charged separately from the transactional store storage and throughput.

The autosync process also takes care of schema updates to the schematized analytical store automatically for you as unique new properties are added over time to items within your container. This allows you to take advantage of the performance advantages provided by schematization without any effort on your part. We will get into more of the details of how analytical store schema is managed and exposed to the Synapse Analytics query capabilities in the next unit.

You can configure the default Time to Live (TTL) property for records stored within the transactional store and analytical store independently of each other. The TTL value of a record defines when it will be automatically deleted from the store. By configuring the default TTL value of both stores, you can manage the lifecycle of data and define how long it will be retained for in each store. You can override the default TTL value (at the item level) for the transactional store however the default TTL value will always apply to data in the analytical and cannot be overwritten at the item level.

Azure Cosmos DB support global distributed accounts replicating your containers transparently to the Azure regions choose. When enabled on a container the analytical store will automatically be configured in all chosen global distribution regions, you cannot selectively choose which regions to deploy an analytical store. It is also recommended that you choose and configure your global distribution regions on the account prior to enabling analytical store on a container.

Understand well defined schemas

There are two constraints that apply to the schema inferencing done by the autosync process as it transparently maintains the schema in the analytical store based on items added or updated in the transactional store:

- You can have a maximum of unique 1000 properties at any nesting level within the items stored in a transactional store. Any property above this and its associated values will not be present in the analytical store.
- Property names must be unique when compared in a case insensitive manner. For example, the properties {"name": "Franklin Ye"} and {"Name": "Franklin Ye"} cannot exist at the same nesting level in the same item or different items within a container given that "name" and "Name" are not unique when compared in a case insensitive manner.

There are two modes of schema representation for data stored in the analytical store. These modes have tradeoffs between the simplicity of a columnar representation, handling the polymorphic schemas, and simplicity of query experience:

- Well-defined schema representation
- Full fidelity schema representation

For SQL (Core) API accounts, when analytical store is enabled, the default schema representation in the analytical store is well-defined. Whereas for Azure Cosmos DB API for MongoDB accounts, the default schema representation in the analytical store is full fidelity schema representation. (If you have scenarios requiring a different schema representation than the default offering for each of these APIs, reach out to the Azure Cosmos DB team to enable it.)

Well-defined schema representation

The well-defined schema representation creates a simple tabular representation of the schema-agnostic data in the transactional store as it copies it to the analytical store.

The following code fragment is an example JSON document representing a customer profile record:

```
{  
    "id": "54AB87A7-BDB9-4FAE-A668-AA9F43E26628",  
    "type": "customer",  
    "name": "Franklin Ye",  
    "customerId": "54AB87A7-BDB9-4FAE-A668-AA9F43E26628",  
    "address": {  
        "streetNo": 15850,  
        "streetName": "NE 40th St.",  
        "postcode": "CR30AA",  
    }  
}
```

The well-defined schema representation has the top-level properties of the documents exposed as columns when queried from both Synapse SQL and Synapse Spark, along with column values that representing the property values, except in the case where those values are of object type or array type, in which case a JSON representation of the properties values contained within are assigned to the column values, and have the following additional considerations:

1. A property always has the same type across multiple items.

For example, {"postcode": 98065} and {"postcode": "CR30AA"} does not have a well-defined schema because "postcode" is sometimes a string and sometimes a number. In this case, the analytical store registers the data type of the postcode attribute within the analytical store, as the data type of the postcode property on its first occurrence in the lifetime of the container, the items where the data type of the postcode attribute differs from this registered value will not be included in the analytical store.

2. This condition does not apply for null properties.

For example, {"postcode": 98065} {"postcode": null} is still well-defined.

3. Array types must contain a single repeated type.

For example, {"postcode": [98065, "CR30AA"]} is not a well-defined schema because the array contains a mix of integer and string types.

Importantly when the Azure Cosmos DB analytical store is using well-defined schema representation mode the data stored in documents added to the container that violates the above rules, these documents will not be included in the analytical store.

When queried from Azure Synapse Analytics, the analytical store record sample will look similar to the following:

Id	Type	Name	CustomerId	Address
54AB87A7-BDB9-4FAE-A668-AA...	customer	Franklin Ye	54AB87A7-BDB9-4FAE-A668-AA...	{"streetNo":15850,"streetName":"NE 40th St.", "postcode":98052}

The following is the result set as shown in Spark:

id	type	name	customerId	address ↑
54AB87A7-BDB9...	customer	Franklin Ye	54AB87A7-BDB9...	▼ {"streetNo":15850,"streetName":"NE 40th St.", "post..." streetNo: "15850" streetName: "NE 40th St." postcode: "98052"

Full-fidelity schema representation

The full-fidelity schema representation creates a more complex tabular representation of the schema-agnostic data in the transactional store as it copies it to the analytical store. The full-fidelity schema representation has the top-level properties of the documents exposed as columns when queried from both Synapse SQL and Synapse Spark along with a JSON representation of the properties values contained within as column values. This is extended to include the data type of the properties along with their property values and as such can better handle polymorphic schemas of operational. With this schema representation, no items are dropped from the analytical store due to the need to meet the well-defined schema rules.

For example, let's take the following sample document in the transactional store:

```
{
  "id": "54AB87A7-BDB9-4FAE-A668-AA9F43E26628",
  "type": "customer",
  "name": "Franklin Ye",
  "customerId": "54AB87A7-BDB9-4FAE-A668-AA9F43E26628",
  "address": {
    "streetNo": 15850,
    "streetName": "NE 40th St.",
    "postcode": "CR30AA",
  }
}
```

The leaf property streetNo within the nested object address will be represented in the analytical store schema as "streetNo":{int32":15850} JSON. The datatype is added as a suffix to the embedded JSON for the address column.

This way, if another document is added to the transactional store where the value of leaf property streetNo is "05851" (note it's a string), the schema of the analytical store automatically evolves without the need to conform to the previously written property types. The leaf property streetNo within the nested address will be represented in the analytical store as "streetNo":{string":05851} JSON.

```
{
  "id": "54AB87A7-BDB9-4FAE-A668-AA9F43E26629",
  "type": "customer",
  "name": "Franklin Ye",
  "customerId": "54AB87A7-BDB9-4FAE-A668-AA9F43E26629",
  "address": {
    "streetNo": "05851",
    "streetName": "NE 40th St.",
    "postcode": 98052
  }
}
```

The following is the resultset shown in Spark:

_id	type	name	customerId	address
["string":"54AB87A7-BDB9-4FAE-A668	▼ ["string":"customer"] string: "customer"	▼ ["string":"Franklin Ye"] string: "Franklin Ye"	▼ ["string":"54AB87A7-I	▼ ["object":{"streetNo":15850,"streetName": "..."} object: {"streetNo":15850,"streetName": "..."} ▼ streetNo: ("int32":15850) int32: "15850" ▼ streetName: ("string": "NE 40th St.") string: "NE 40th St." ▼ postcode: ("int32":98052) int32: "98052"
► ["string":"54AB87A7-BDB9-4FAE-A668	▼ ["string":"customer"] string: "customer"	▼ ["string":"Franklin Ye"] string: "Franklin Ye"	▼ ["string":"54AB87A7-B	▼ ["object":{"streetNo":05851,"streetNa..."} object: {"streetNo":05851,"streetName": "str..."} ▼ streetNo: ("string":05851) string: "05851" ▼ streetName: ("string": "NE 40th St.") string: "NE 40th St." ▼ postcode: ("string":CR30AA) string: "CR30AA"

The following is the resultset shown in SQL:

_id	Type	Name	Customerid	Address
["string":"54AB87A7-BD...	["string":"customer"]	["string":"Franklin Ye"]	["string":"54AB87A7-BDB9-4FAE-A668-A...	["object":{"streetNo":15850,"streetName": "..."} object: {"streetNo":15850,"streetName": "..."} ▼ streetNo: ("int32":15850,"streetName": "...") int32: "15850" ▼ streetName: ("string": "NE 40th St.") string: "NE 40th St." ▼ postcode: ("int32":98052) int32: "98052"
["string":"54AB87A7-BD...	["string":"customer"]	["string":"Franklin Ye"]	["string":"54AB87A7-BDB9-4FAE-A668-A...	["object":{"streetNo":05851,"streetName": "str..."} object: {"streetNo":05851,"streetName": "str..."} ▼ streetNo: ("string":05851) string: "05851" ▼ streetName: ("string": "NE 40th St.") string: "NE 40th St." ▼ postcode: ("string":CR30AA) string: "CR30AA"

For reference here is a map of all the property data types and their suffix representations in the analytical store along with an indication of the API where these property data types are supported

Original data type	SQL (Core) API	MongoDB API	Suffix name	Example
Double	X	X	".float64"	24.99
Array	X	X	".array"	["a", "b"]
Binary		X	".binary"	0
Boolean	X	X	".bool"	TRUE
Int32	X	X	".int32"	123
Int64	X	X	".int64"	255486129307
Null	X	X	".null"	null
String	X	X	".string"	"ABC"
Timestamp		X	".timestamp"	Timestamp(0, 0)
DateTime		X	".date"	ISODate("2020-08-21T07:43:07.375Z")
ObjectId		X	".objectId"	ObjectId("5f3f-7b59330ec-25c132623a2")
Document	X	X	".object"	{"a": "a"}

Design schemas to support type of analytics

Now that you understand how the data is transformed and represented in the analytical store, there are some additional considerations that you would wish to take into account when you model your data to enable it to more comfortably support analytical queries of the analytical store.

Mixed entity types per container

You may want to mix different document entity types (entities) in the same container, which is useful to efficiently retrieve data for both entities using a single query. For example, you could put both customer profile and sales order data in the same container and partition it by customerId. In such a situation, you would usually add a field to your documents that identifies the entity type of each document to differentiate between them at query time. In the following sample documents, you will see that the type is added for this purpose in the following example documents:

```
{
  "id": "54AB87A7-BDB9-4FAE-A668-AA9F43E26628",
  "type": "customer",
  "name": "Franklin Ye",
  "customerId": "54AB87A7-BDB9-4FAE-A668-AA9F43E26628",
  "address": {
    "streetNo": 15850,
    "streetName": "NE 40th St.",
    "postcode": 98052
  }
}

{
  "_id": "000C23D8-B8BC-432E-9213-6473DFDA2BC5",
  "type": "salesOrder",
  "customerId": "54AB87A7-BDB9-4FAE-A668-AA9F43E26628",
  "orderDate": "2014-02-16T00:00:00",
  "shipDate": "2014-02-23T00:00:00",
  "details": [
    {
      "sku": "BK-R64Y-42",
      "name": "Road-550-W Yellow, 42",
      "price": 1120.49,
      "quantity": 1
    }
  ]
}
```

The following query on against the transactional store would return the customer details and all orders associated with this one customer.

```
SELECT * FROM c WHERE c.customerID = "54AB87A7-BDB9-4FAE-A668-AA9F43E26628"
```

Whilst this approach to modeling is potentially useful for your Cosmos DB transactional store queries. All documents within a single container are mapped to a single analytical store, leading to sparsely populat-

ed column stores with the different data types needing to be further separated at the time of running an analytical query.

Recommendation: As with many design decisions, there is a trade-off between the efficiency of querying the transactional store and the ease of querying the analytical store. Carefully evaluate the usefulness of storing a mix of different document entity types in the same container to your transactional workloads. If you choose to do so, you will be required to filter by the property entity type property you selected.

Embedding entity arrays

When optimizing transactional data models, we choose to embed entities within an array in a document, especially for read heavy workloads where:

- There are contained relationships between entities.
- There are one-to-few relationships between entities.
- There is embedded data that changes infrequently.
- There is embedded data that will not grow without bound.
- There is embedded data that is queried frequently together.

Due to the fact that there are one to few relationships between the embedded entities that are represented within a single document, and that these are mapped to a single column within a single row within the analytical store. The entire embedded entity array will reside within a single column value, and need to be translated from its JSON representation at the time of querying in order to retrieve embedded entity values, irrespective of which of the two modes of schema representation being used.

Recommendation: Again, a balance needs to be struck between the usefulness of the entity embedding within the transactional application and the added complexity of writing queries against embedded JSON documents for your application.

Partitioning of data

All data within an Azure Cosmos DB container is partitioned based on the partition key, and applies to both the transactional store and the analytical store. Boundaries for parallelizing workloads are based on this partition key.

The orderliness associated when data appears in the analytical store for a query is only guaranteed within a partition. As an example, when documents (1) (2) (3) are inserted in the transactional store into a single partition, they are guaranteed to be present in the analytical store in the order in which they were inserted.

Design writeback scenarios

The Azure Cosmos DB analytical store is read-only from the point of view of analytical workloads; however, there are scenarios whereby the results of analytical queries needs to be retrieved by clients that are using the Cosmos DB transactional store.

For these requirements, the results of analytical queries run against a Cosmos DB analytical store, can be written back to the Cosmos DB transactional store using Azure Synapse Apache Spark.

Knowledge check

Question 1

What is the name of the application architecture that enables near real-time querying to provide insights?

- OLTP.
- HTAP.
- OLAP.

Question 2

Within Azure Synapse Link for Azure Cosmos DB, which Column-oriented store optimized for queries?

- Transactional store.
- Cosmos DB store.
- Analytical store.

Question 3

How can you manage the lifecycle of data and define how long it will be retained for in an analytical store?

- Configure the purge duration in a container
- Configure the default Time to Live (TTL) property for records stored.
- Configure the deletion duration for records in the transactional store.

Summary

You have seen how Azure Synapse Analytics Link for Cosmos DB solves the issue of making operational data available for analytical queries in near real time using the Hybrid Transactional and Analytical Processing.

Now that you have completed this module, you have learned:

- Understand Hybrid Transactional and Analytical Processing patterns
- Describe the supported analytical workloads
- Explain HTAP scenarios in the context of Synapse Link
- Describe the analytical store
- Understand well-defined schemas
- Design schemas to support type of analytics
- Design writeback scenarios

Configure Azure Synapse Link with Azure Cosmos DB

Introduction

Learn how to leverage Azure Synapse Analytics Link for Cosmos DB to make operational data stored in Azure Cosmos DB available for analytical query from within Azure Synapse using both SQL and Spark.

In this module, you will:

- Enable Cosmos DB Account to use Azure Synapse Link
- Create an analytical store enabled container
- Implement Synapse Link for Cosmos DB
- Validate connectivity from Spark
- Validate connectivity from SQL Serverless

Prerequisites

Before taking this module, it is recommended that the student is able to:

- Have a good understanding of Azure Cosmos DB capabilities
- Have a working knowledge of SQL or Spark
- Know how to deploy an Azure Synapse Analytics Workspace
- Know how to deploy an Azure Cosmos DB Account

Note: Throughout this module and those that follow we are going to be using two Azure Cosmos DB accounts, one configured for Azure Cosmos DB SQL (Core) API and a second account configured for the Azure Cosmos DB API for MongoDB in order to highlight the differences and similarities of the experience, specifically using their different default schema representation modes.

Enable Cosmos DB account to use Azure Synapse Link

Before we can create an Azure Cosmos DB container with an analytical store, we must first enable Azure Synapse Link on the Azure Cosmos DB account.

Note: Today you cannot disable the Synapse Link feature once it is enabled on the account, you cannot disable it. Enabling Synapse Link on the account has no billing implications until containers are created with the analytical store enabled.

Enabling Synapse Link on Azure Cosmos DB SQL (Core) API account

To enable Azure Synapse link on your previously created Azure Cosmos DB SQL (Core) API preform the following steps:

1. Navigate to the **Azure portal**¹ and select the Azure Cosmos DB account.

¹ <https://portal.azure.com>

2. Navigate to your previously created Azure Cosmos DB SQL (Core) API account

The screenshot shows the Microsoft Azure portal interface. The URL in the address bar is 'Home > Resource groups > SynapseLink-AdventureWorks > adventureworks-sql'. The left sidebar has 'Data Explorer' selected (marked with a red circle and the number 3). At the top of the main content area, there is a button labeled 'Enable Azure Synapse Link (Preview)' (marked with a red circle and the number 4).

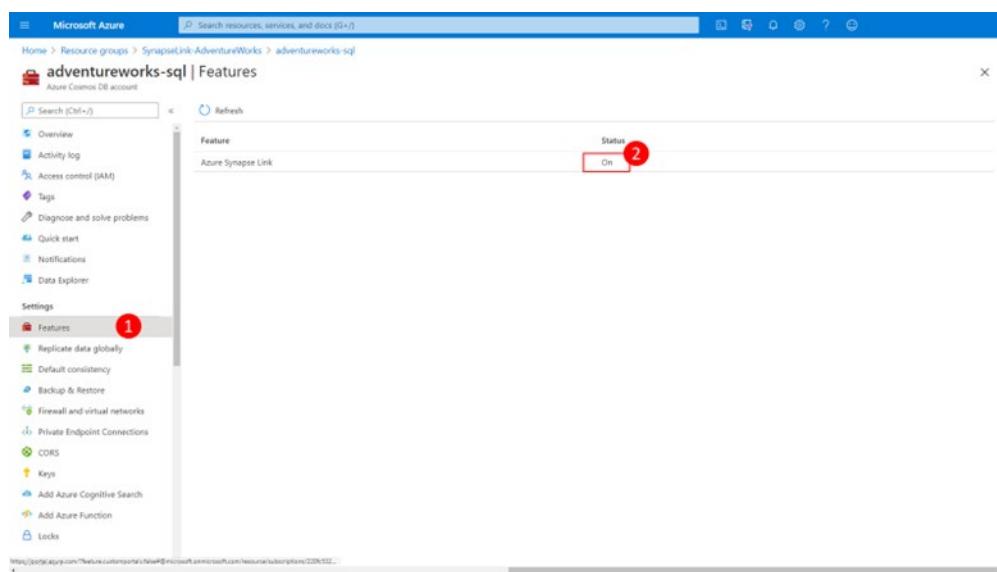
3. Select **Data Explorer** in the left-hand menu (3)
4. Then click **Enable Azure Synapse Link** button at the top of the screen (4)

The screenshot shows the Microsoft Azure portal interface with the same account 'adventureworks-sql'. A pop-up dialog box is centered on the screen, asking 'Enable Azure Synapse Link on your Cosmos DB account?'. The 'Enable Azure Synapse Link' button is highlighted with a red circle and the number 5.

5. Then click **Enable Azure Synapse Link** on the pop-up dialog box.

Within a few minutes Azure Synapse Link will be enabled on the account.

To verify that the Azure Synapse Link feature is enabled on the account follow the follow steps:

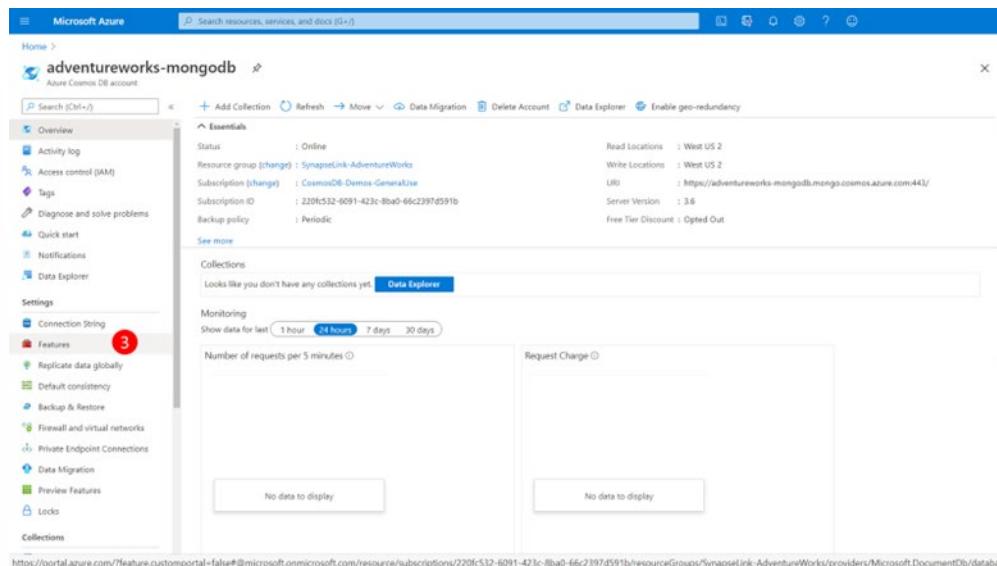


6. Select **Features** in the left-hand menu (1)
7. Verify that the Azure Synapse Link feature shows with a Status of **on** (2), this will indicate that the Azure Cosmos DB account has been enabled for Azure Synapse Link.

Enabling Synapse Link on Azure Cosmos DB API for MongoDB account

We will now use an alternative and equally effective manner for enabling the Synapse Link on your previously provisioned Azure Cosmos DB API for MongoDB account

1. Navigate to the Azure portal (<https://portal.azure.com>) and select the Azure Cosmos DB account.
2. Navigate to your previously created Azure Cosmos DB API for MongoDB account



3. Select **Features** in the left-hand menu (3)

Feature	Status
Azure Synapse Link	Off

4. Then click on the **Azure Synapse Link** entry in the features table (4).

5. The click the **Enable** button on the dialog box on the right (5)

You will then see a notification pop-up that letting you know that Azure is Enabling Synapse Link and will complete within a few minutes.

You can verify that your Azure Cosmos DB API for MongoDB account has been enabled for with the Synapse Link feature in the same manner we did for the Azure Cosmos DB Core (SQL) API account previously.

Create an analytical store enabled container

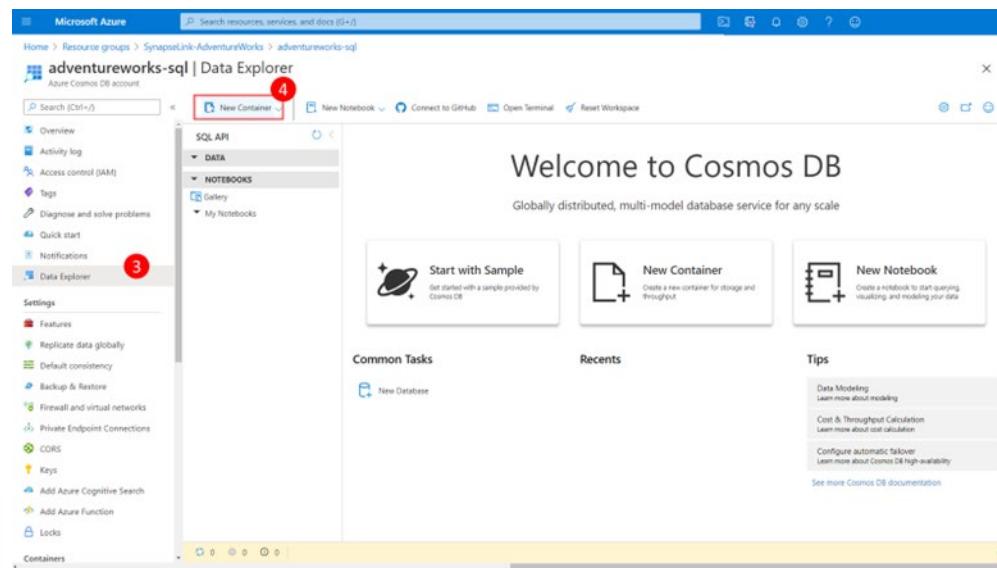
Before we can query our data using Azure Synapse Analytics using Azure Synapse Link, we must first create the container that is going to hold our data at the same time enabling it to have an analytical store.

Note: Today enabling analytical store is only available at the time of creating a container and cannot be completely disabled without deleting the container. Setting the default analytical store TTL value to 0 or null effectively disables the analytical store by no longer synchronize new items to it from the transactional store and deleting items already synchronized from the analytical store.

Create a new Azure Cosmos DB Core (SQL) API container

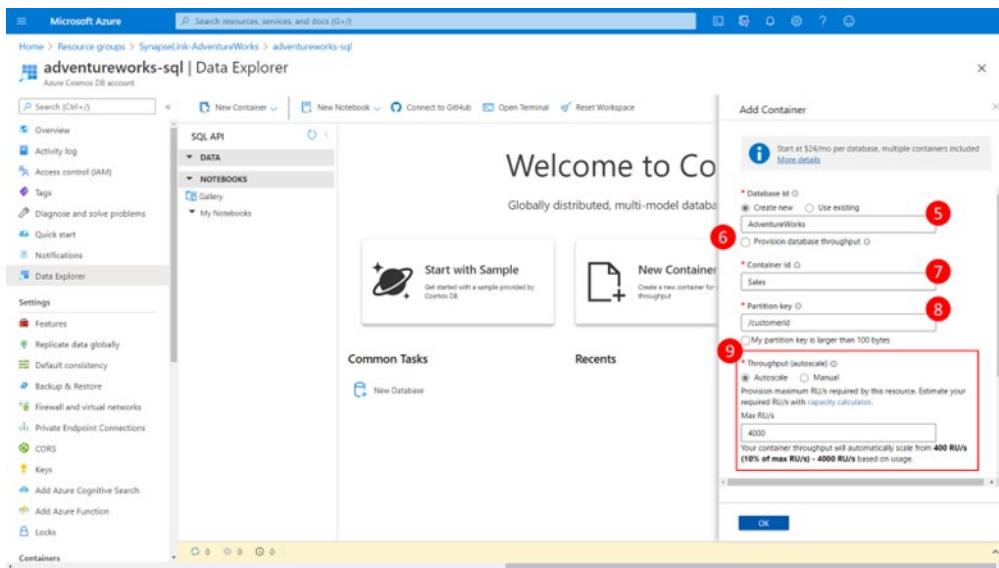
To create a new Azure Cosmos DB Core (SQL) API container with analytical store enabled, follow the following steps:

1. Navigate to the Azure portal [Azure portal²](https://portal.azure.com) and select the Azure Cosmos DB account.
2. Navigate to your previously created Azure Cosmos DB Core (SQL) API account
3. Select **Data Explorer** on the left-hand menu (3).
4. Click the **New Container** button at the top of screen.

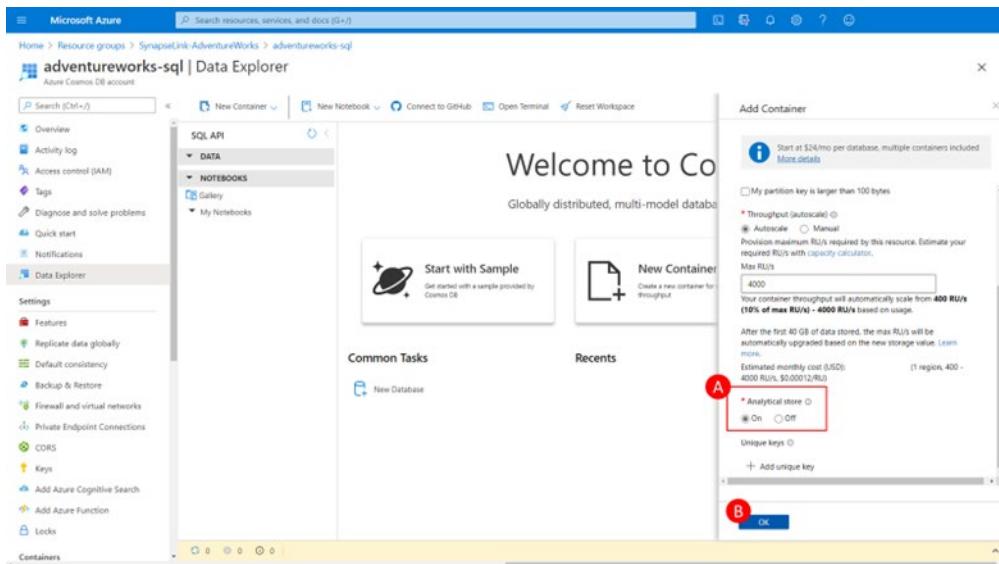


An Add Container dialog will appear.

² <https://portal.azure.com>



5. Enter the **new databases** and **container** information:
 - a. For the **Database ID** we choosing to use AdventureWorks (5)
 - b. Ensure that you unselect the Provision database throughput checkbox (6)
 - c. For the **container ID** we choose Sales
 - d. Enter /customerID for the **Partition Key**(8)
6. Choose the throughput for your container by selecting **Autoscale** and **specify a Max RU/s of 4000** (9)



7. Scroll down and enable analytical store by ensuring that the **Analytical store** on radio button is selected (A)
8. Click **OK** to create the container.

Whilst this module assumes you understand how to appropriately configure an Azure Cosmos DB container to maximize performance and minimize cost lets briefly go over some of the thinking used to choose the values we did:

- We choose a partition key property of customerID as this attribute is used in many of the queries used to retrieve customer and sales order information in our application, it has relatively high cardinality (number of unique values) and thus will allow our container to scale as the number of customers and sales orders grows.
- We chose to use autoscale provisioned throughput and set the maximum value to 4000 RU/s as we are just starting with our application and don't expect massive query volumes initial. A max value 4000 RU/s will enable the container to automatically scale between this value all the way down to 10% of this max value (400 RU/s) when not needed. This should be plenty throughput for what we are going to do today.

Load sample data into the Azure Cosmos DB Core (SQL) API container

Perform the following steps to load a couple of sample items into the newly created container:

1. Navigate to the Azure portal [Azure portal³](https://portal.azure.com) and select the Azure Cosmos DB account.
2. Navigate to your previously created Azure Cosmos DB Core (SQL) API account
3. Select **Data Explorer** in the left-hand menu **(3)**
4. Navigate to the **Items folder** with the Sale container we just created by:
 - a. Expanding AventureWorks database **(4)**
 - b. Expanding the Sale Container **(5)**
 - c. Clicking on the **Items** folder **(6)**
- You will see no item listed in the items list **(7)**; the container is empty.
5. Now create a new customer profile item by:
 - a. Clicking the **New Item** button on the top ribbon **(7)**
 - b. Enter customer profile JSON in the **edit pane** **(8)**
 - c. Click the **Save** button on the ribbon **(9)** to save the item.

³ <https://portal.azure.com>

The screenshot shows the Microsoft Azure Data Explorer interface for the 'adventureworks-sql' database. The left sidebar shows various database settings and features. The main area displays the 'SQL API' interface with the 'DATA' section selected. Under 'AdventureWorks', the 'Sales' collection is shown. A red circle labeled 'A' highlights a new row in the 'Items' list pane. A red circle labeled 'B' highlights the JSON editor pane where the newly added item's details are visible.

```

1 "id": "544887A7-BD89-4FAE-A668-AA9F43E26428",
2 "type": "Customer",
3 "name": "Franklin Ya",
4 "customerId": "544887A7-BD89-4FAE-A668-AA9F43E26428",
5 "addressId": "544887A7-BD89-4FAE-A668-AA9F43E26428",
6 "street1": "15800",
7 "street2": "NE 40th St.,",
8 "postCode": "98052"
9
10
11 "-ext": "wP0T35jyR8AAAAAAAAB+-",
12 "-ext": "dbs/wP0T35jyR8AAAAAAAAB+-/colls/wP0T35jyR8AAAAAAAAB+-/+",
13 "-tag": "{\"42200000-0000-0000-0000-5f0e1a500000\":}",
14 "-t": "Customer",
15 "-x": 1606291881

```

You will now see a new row in the items list (A), and if you click on this row in the items list an updated version of the item will appear in the items pane, that includes some additional item meta data the service automatically adds and maintains within the item body when an item is added or updated (B)

6. Now create a new sales order item for our previously added customer by:
 - a. Clicking the New Item button on the top ribbon (C)
 - b. Copy the customer sales order JSON from below and past it in the edit pane (D)
 - c. Click the Same button on the ribbon (9) to save the item.

The screenshot shows the Microsoft Azure Data Explorer interface for the 'adventureworks-sql' database. The left sidebar shows various database settings and features. The main area displays the 'SQL API' interface with the 'DATA' section selected. Under 'AdventureWorks', the 'Sales' collection is shown. A red circle labeled 'C' highlights the 'New Item' button on the ribbon. A red circle labeled 'D' highlights the JSON editor pane where the sales order item is being typed. A red circle labeled 'E' highlights the 'Save' button on the ribbon.

```

1 "id": "000K230E-888C-432E-9213-64730FDA28C5",
2 "type": "SalesOrder",
3 "customerId": "544887A7-BD89-4FAE-A668-AA9F43E26428",
4 "orderDate": "2014-02-17T00:00:00Z",
5 "shipDate": "2014-02-23T00:00:00Z",
6 "details": [
7     {
8         "sku": "BK-BABY-42",
9         "name": "Road-550-W Yellow, 42",
10        "price": 1120.49,
11        "quantity": 1
12    }
13 ]

```

And you will now see a second document in the Items list (F)

The screenshot shows the Microsoft Azure Data Explorer interface for the 'adventureworks-sql' database. The left sidebar shows various database settings and a 'Data Explorer' section. The main area displays a query results pane with a red circle labeled 'F' pointing to the 'New SQL Query' icon. A second red circle labeled 'G' points to the query window where the following SQL code is entered:

```
SELECT * FROM c WHERE c.customerId = '54AB87A7-BD89-4FA...
```

7. Let run a quick query against this data to retrieve customer profile and sales order information for a specific customer, this is typical of the queries our application runs:
 - a. Click the **New SQL Query** icon (**F**)
 - b. Enter the query into the query window (**G**)
 - c. Click the **Execute** button (**H**)

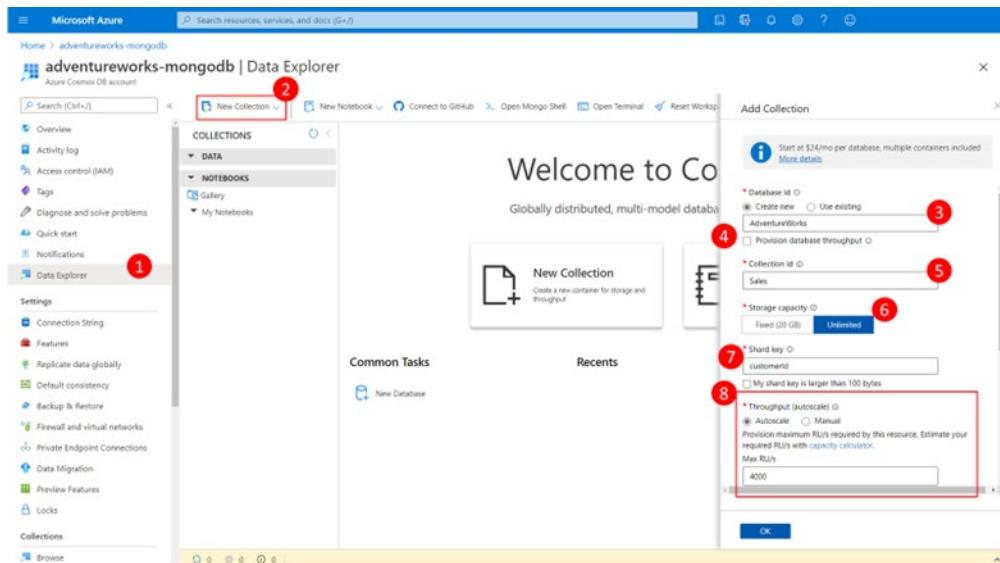
You will see the results immediately returned in the results pain including the content of both items we previously created (**I**)

The screenshot shows the Microsoft Azure Data Explorer interface for the 'adventureworks-sql' database. The left sidebar shows various database settings and a 'Data Explorer' section. The main area displays a query results pane with a red circle labeled 'F' pointing to the 'New SQL Query' icon, another red circle labeled 'G' pointing to the query window containing the same SQL code as before, and a third red circle labeled 'H' pointing to the 'Execute Query' button. Below the results pane, a message says 'Execute a query to see the results'. The results pane itself is currently empty.

Create a new Azure Cosmos DB API for MongoDB container

To create a new Azure Cosmos DB API for MongoDB container with analytical store enabled by executing the following steps, in a manner similar to what we recently did for the SQL API

1. Navigate to the Azure portal [Azure portal](https://portal.azure.com)⁴ and select the Azure Cosmos DB account.
2. Navigate to your previously created Azure Cosmos DB API for MongoDB account

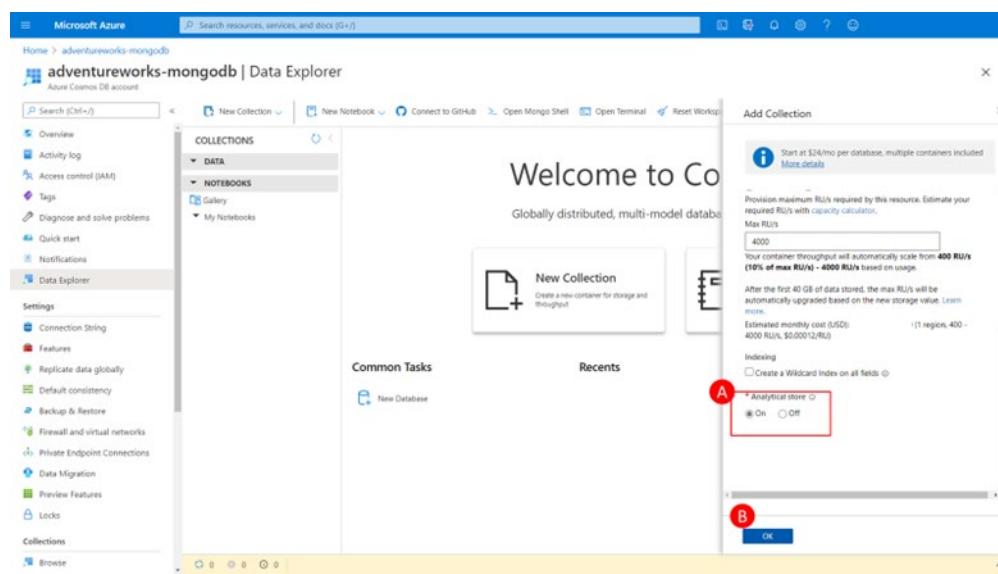


3. Select **Data Explorer** on the left-hand menu (1).
4. Click the **New Container** button at the top of screen (2).

An Add Container dialog will appear.

5. Enter the new databases and container information:
 - a. For the **Database ID** we choose to use AdventureWorks (3)
 - b. Ensure that use unselect the **Provision database throughput** checkbox (4)
 - c. For the **container ID** we choose Sales (4)
 - d. Select that we want unlimited storage capacity (6), this is an option that we don't have on creating a SQL API container since all SQL API containers are now unlimited.
 - e. Enter customerID for the **Shard Key** (7), this is the MongoDB API equivalent of the SQL API partition key.
6. Choose the throughput for your container by selecting **Autoscale** and specify a Max RU/s of 4000 (8)

⁴ <https://portal.azure.com>

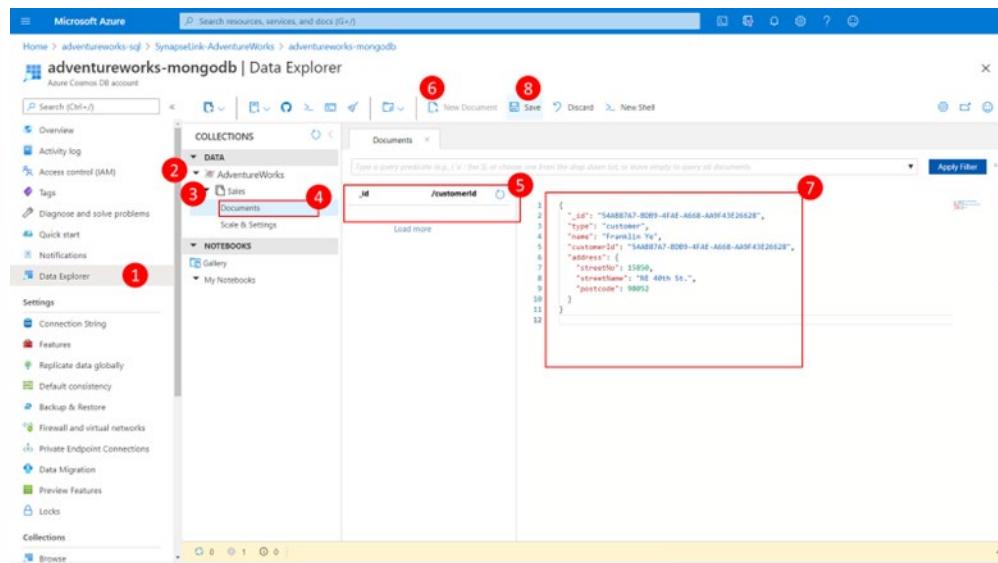


7. Scroll down and enable analytical store by ensuring that the **Analytical store** on radio button is selected (A)
8. Click **OK** to create the container (B).

Load sample data into the Azure Cosmos DB API for MongoDB container

Perform the following steps to load a couple of sample items into the newly created MongoDB API container:

1. Navigate to the Azure portal [Azure portal](#)⁵ and select the Azure Cosmos DB account.
2. Navigate to your previously created Azure Cosmos DB API for MongoDB account



⁵ <https://portal.azure.com>

3. Select **Data Explorer** in the left-hand menu (1)
4. Navigate to the items folder with the Sale container we just created by:
 - a. Expanding AventureWorks database (2)
 - b. Expanding the Sale Collection (3)
 - c. Clicking on the Documents folder (4)

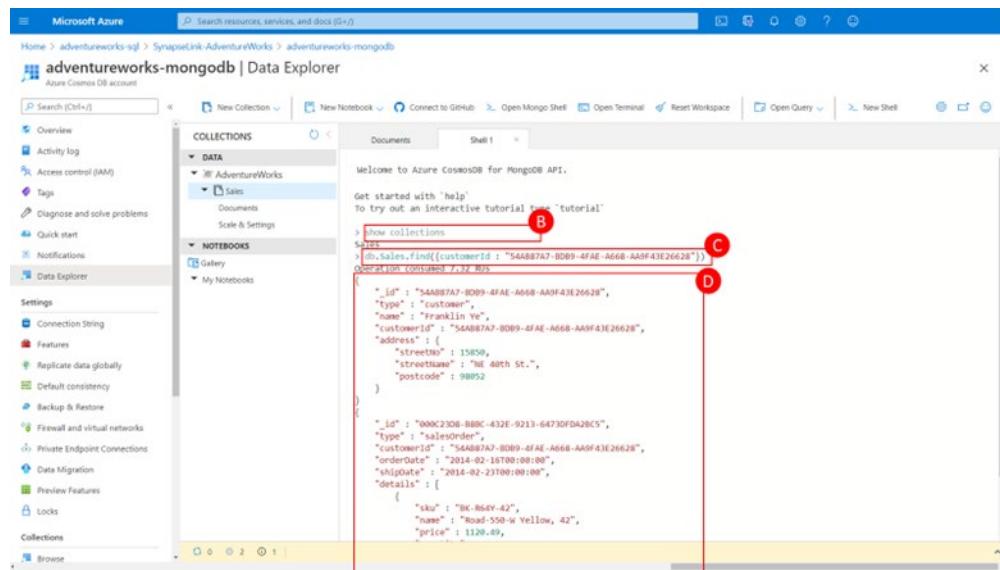
You will see no Document _IDs listed in the documents list (5); the collection is empty.
5. Now create a new customer profile item by:
 - a. Clicking the New Document button on the top ribbon (6)
 - b. Enter code into the edit pane (7)
 - c. Click the **Save** button on the ribbon (8) to save the item.

You will now see a new row in the documents list.
6. Now create a new sales order document for our previously added customer by:
 - a. Click the **New Document** button on the top ribbon (6)
 - b. Enter code into the edit pane.
 - c. Click the **Save** button on the ribbon (8) to save the item.

And you will now see a second document in the document list.
7. Let's run a quick query against this data to retrieve customer profile and sales order information for a specific customer, this is typical of the queries our application runs:

ID	/customerId
54A8E7A7-BD89-4FA...	54A8E7A7-BD89-4FA...
00C2208-888C...	54A8E7A7-BD89-4FA...

- a. Click the New Shell (A)



This will open a mongo client shell within the portal.

- b. Type "show collections" and press enter **(B)**

This will return the Sales collection as the only collection in our database.

- c. Type cmd db.Sales.find({“customerID” : “54AB87A7-BDB9-4FAE-A668-AA9F43E26628”} and press enter **(C)**

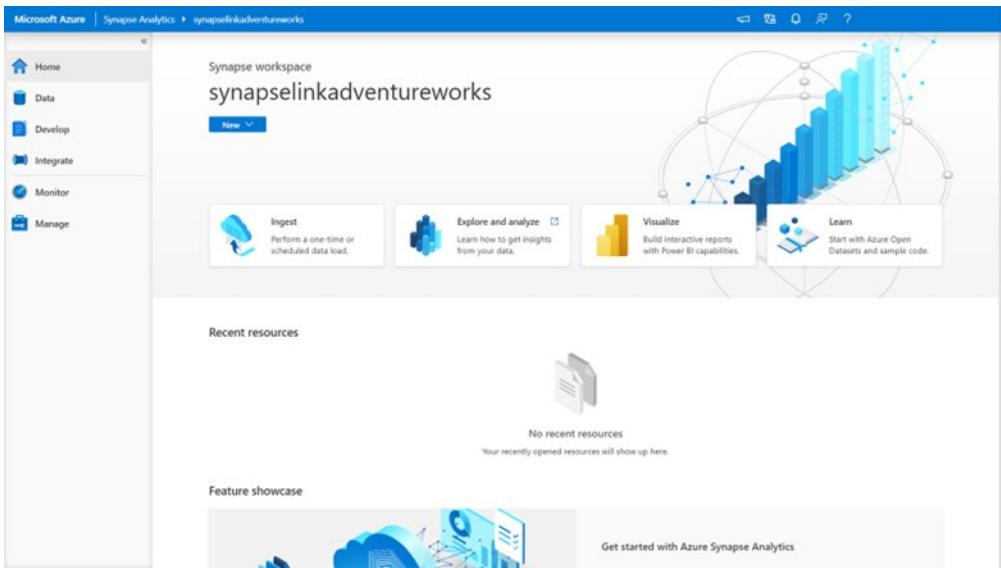
This will return our results immediately including the content of both documents we just created **(D)**

Implement Synapse Link for Cosmos DB

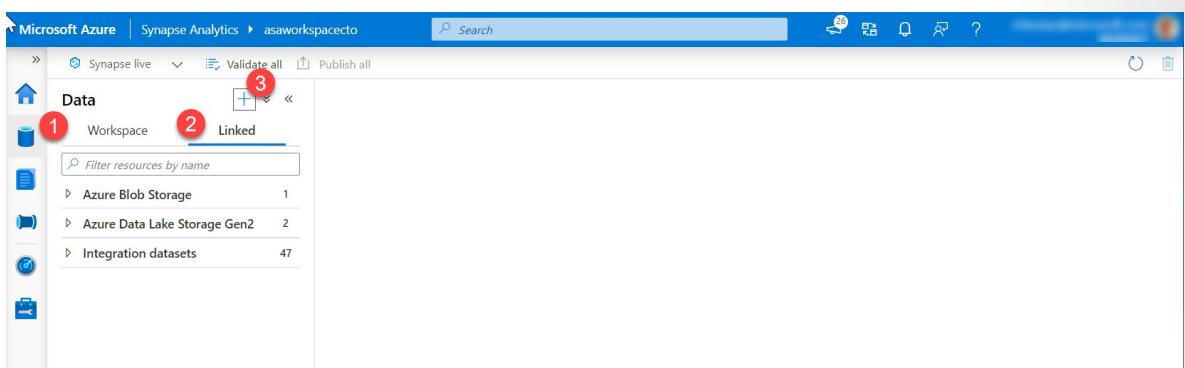
In order to query the data within our Cosmos DB analytical store from Azure Synapse using Spark, we need to configure a linked service

Configure Azure Synapse Linked Service for Azure Cosmos DB Core (SQL) API

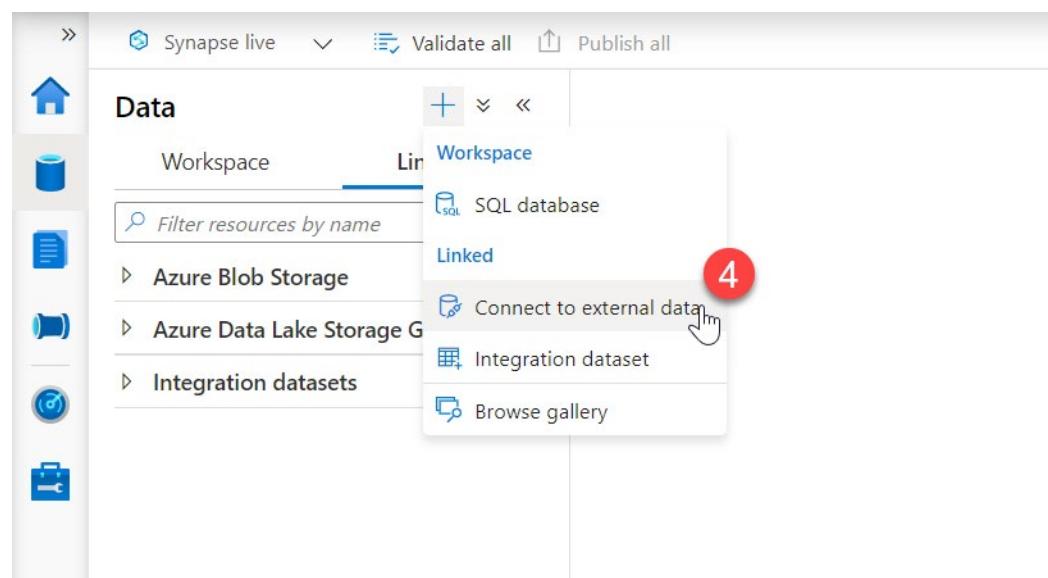
To configure the Azure Synapse Linked Service for Azure Cosmos DB Core (SQL) API, perform the following steps:



1. Connect to a previously deployed Azure Synapse Workspace running an Azure Synapse SQL Serverless instance (**deployed by default with the workspace**) and an Azure Synapse Spark Pool (**you need to have previously deployed this**).



2. In the left-hand menu, select **Data** (1)
3. Click on the **Linked tab** in the explorer view (2)
4. Click the **+** button to add a resource (3)



5. Select **Connect to external data** from the list that pops up (4)

Connect to external data

Once a connection is created, the underlying data of that connection will be available for analysis in the Data hub or for pipeline activities in the Integrate hub.

A screenshot of the 'Connect to external data' dialog box. It displays a grid of six connection options:

- Azure Blob Storage
- Azure Cosmos DB (MongoDB API)
- Azure Cosmos DB (SQL API) (highlighted with a red circle labeled 5)
- Azure Data Explorer (Kusto)
- Azure Data Lake Storage Gen1
- Azure Data Lake Storage Gen2

At the bottom of the dialog, there are 'Continue' and 'Cancel' buttons, with a red circle labeled 6 next to the 'Continue' button.

6. Select **Azure Cosmos DB SQL API**

7. Click **Continue**

New linked service (Azure Cosmos DB (SQL API))

Choose a name for your linked service. This name cannot be updated later.

Name * 7

Description

Connect via integration runtime * 8

Connection string Azure Key Vault 9

Account selection method A

From Azure subscription Enter manually

Azure subscription 8

Azure Cosmos DB account name * 9

Database name * A ↻

Additional connection properties + New

Annotations + New

Name C

Parameters D

Advanced E

B C D E

Create Back Test connection Cancel

8. Type **AdventureWorksSQL** as the name of our linked service (7)
9. Select the **Azure Subscription** in which the previously configured Azure Cosmos DB Core (SQL API) account exists from the drop-down list (8)
10. Select the **Azure Cosmos DB account** from the drop-down list (9)
11. Select the **Sales** database for the drop-down list

12. Click **Test connection**

You should immediately receive confirmation that the connection was successful.

13. Click **Create**

You are done creating your Linked service for your Azure Cosmos DB Core (**SQL**) API

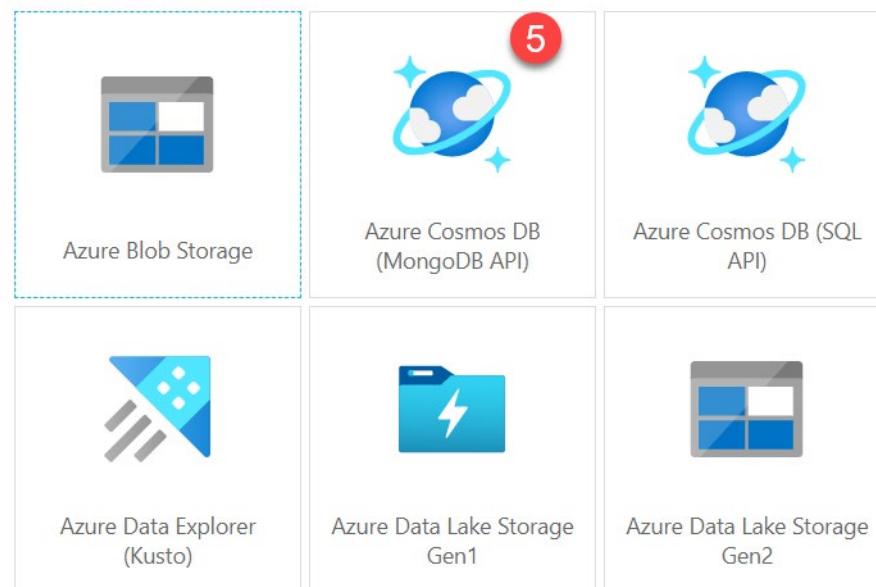
Configure Azure Synapse linked service for Azure Cosmos DB API for MongoDB

To configure the Azure Synapse Linked Service for Azure Cosmos DB API for MongoDB, perform the following steps:

1. Connect to a previous deployed Azure Synapse Workspace
2. In the left-hand menu, select **Data**
3. Click on the **Linked tab** in the explorer view.
4. Click the **+ button** to add a resource.
5. Select **Connect to external data** from the list that pops up.

Connect to external data

Once a connection is created, the underlying data of that connection will be available for analysis in the Data hub or for pipeline activities in the Integrate hub.



6. Select **Azure Cosmos DB (MongoDB API) (5)**

7. Click **Continue (6)**

8. Type **AdventureWorksMongoDB** as the name of our linked service

9. Select the **Azure Subscription** in which the previously configured Azure Cosmos DB API for MongoDB account exists from the drop-down list.

10. Select the **Azure Cosmos DB account** from the drop-down list

11. Select the **Sales** database for the drop-down list

12. Click **Test connection**

You should immediately receive confirmation that the connection was successful.

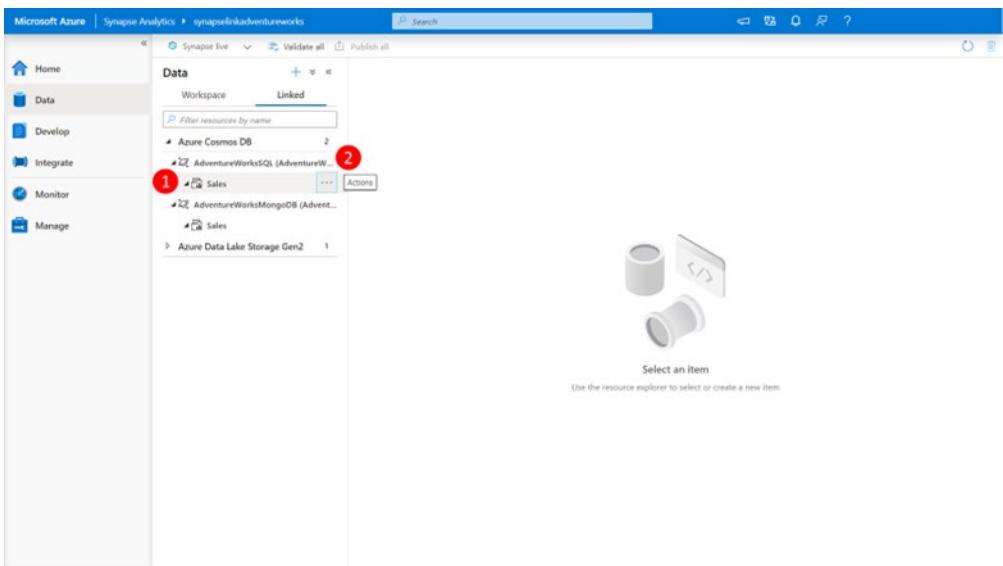
13. Click **Create**

You are done creating your Linked service for your Azure Cosmos DB API for MongoDB

Validate connectivity from Spark

Spark Queries for Cosmos DB Core (SQL) API

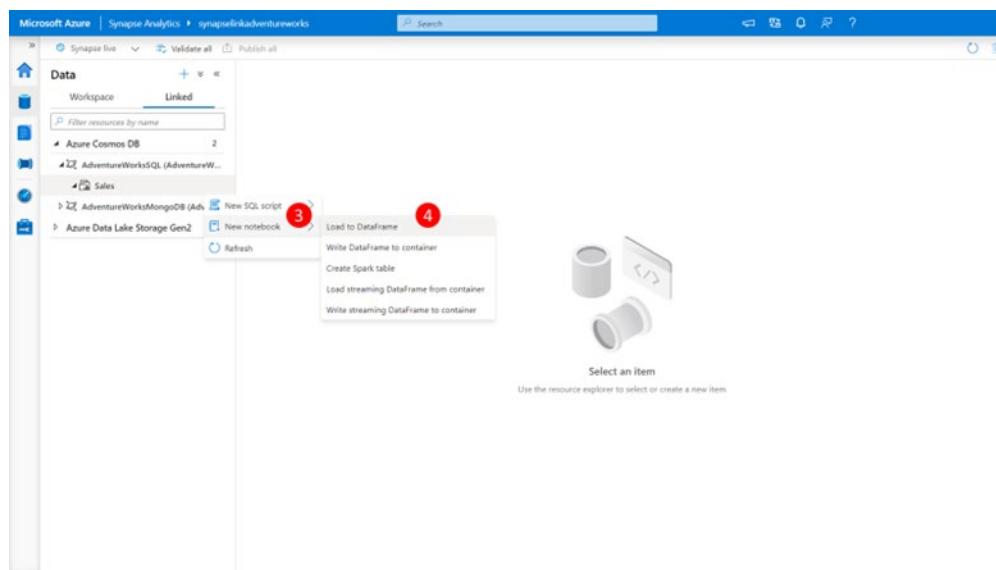
Let's now connect to our Cosmos DB Cosmos DB Core (SQL) API analytical store using Spark and retrieve some data by performing the following steps:



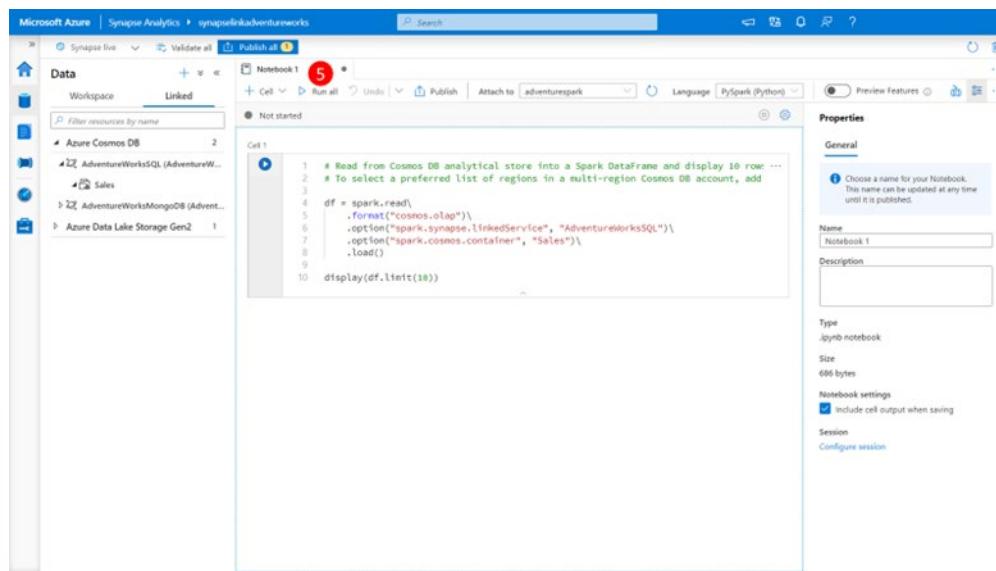
1. Expand the **AdventureWorksSQL linked service** in the explorer view and click on the **Sale container**

(1)

2. Click on the **Actions ellipsis “...”**



3. Click on **new notebook** to expose the list of notebooks actions.
4. Click on **Load to DataFrame**, to load a prepopulated notebook with a spark query to retrieve the top 10 records from the Linked Server and its associated analytical store.



5. Click the **Run All** button on the ribbon to execute the notebook.

The screenshot shows a Microsoft Azure Synapse Analytics notebook interface. The left sidebar displays connected data sources: Azure Cosmos DB, AdventureWorksSQL, and AdventureWorksMongoDB. The notebook cell contains the following PySpark code:

```

1 # Read from Cosmos DB analytical store into a Spark DataFrame and display 10 rows from the DataFrame
2 # To select a preferred list of regions in a multi-region Cosmos DB account, add .option("spark.cosmos.preferredRegion"
3
4 df = spark.read
5   .format("cosmos.olap")
6   .option("spark.synapse.linkedService", "AdventureworksSQL")
7   .option("spark.cosmos.container", "Sales")
8   .load()
9
10 display(df.limit(10))

```

The output pane shows the execution status: "Job execution Succeeded" with "Spark 2 executors 16 cores". Below this, the results are displayed in a table view. A red circle labeled (6) points to the table header. A red circle labeled (7) points to the "name" column for the salesOrder record, which contains a null value. Another red circle labeled (8) points to the "address" column for the customer record, which contains a JSON object.

You should almost immediately see the query begin to execute and the shortly thereafter receive back a result set (6)

Note: That for records where data was not defined, such as the name column for the salesOrder record we get back a null value.

This screenshot shows the same notebook environment as the previous one. The code and execution details are identical. The results table now highlights two specific columns: "address" (red circle 8) and "details" (red circle 9). The "address" column for the customer record contains a JSON object with street information. The "details" column for the same record contains a JSON array with a single element, representing a road price detail.

6. Scrolling right though the result set

You will see that columns that contain JSON objects, such as address (8) and JSON arrays such a detail (9) has the JSON as their column content.

Spark queries for Azure Cosmos DB API for MongoDB

Let's now connect to our Azure Cosmos DB API for MongoDB analytical store using Spark and retrieve some data by performing the following steps:

1. Expand the **AdventureWorksMongoDB linked service** in the explorer view and click on the **Sale container**.

2. Click on the **Actions ellipsis “...”**
3. Click on **new notebook** to expose the list of notebooks actions.
4. Click on **Load to DataFrame**, to load a prepopulated notebook with a spark query to retrieve the top 10 records from the Linked Server and its associated analytical store.

```

1 # Read from Cosmos DB analytical store into a Spark DataFrame and display 10 rows
2 # To select a preferred list of regions in a multi-region Cosmos DB account, add
3 # .option("spark.cosmos.preferredRegion", "[YourLinkedServiceName]")
4 df = spark.read(
5   .format("cosmos.olap")
6   .option("spark.synapse.linkedService", "AdventureworksMongoDB")
7   .option("spark.cosmos.container", "Sales")
8   .load()
9 )
10 display(df.limit(10))

```

5. Update the linked service name and click the **Run All** button on the ribbon to execute the notebook.

_id	_s	id	_etag	_jd	type	name	customerId	address
1fg1APSVy4BD...	1606293078	NTRBQij3QTH...	"08009336-0000..."	► {"string": "544887"}	► {"string": "customer"}	► {"string": "Franklin"}	► {"string": "544887"}	► {"object": {}}
1fg1APSVy4BEA...	1606293181	MDAwQztlEDgt...	"08009456-0000..."	► {"string": "000C23"}	► {"string": "salesOrder"}	► {"string": "544887"}		

You should almost immediately see the query begin to execute and the shortly thereafter receive back a result set (6)

Note: That for records where data was not defined, such as the name column for the salesOrder record we get back a null value, and that the name column now contains a JSON fragment with both

the data type and value since the MongoDB API uses full fidelity schema mode by default.

```

Microsoft Azure | Synapse Analytics - synapsefulladventureworks
Data + v < Publish all 1
Workspace Linked
Filter resources by name
Azure Cosmos DB 2
AdventureWorksSQL (AdventureW...
AdventureWorksMongoDB (Advent...
Sales
Azure Data Lake Storage Gen2 1

Cell 1
1 # Read from Cosmos DB analytical store into a Spark DataFrame and display 10 rows from the DataFrame ...
2 # To select a preferred list of regions in a multi-region Cosmos DB account, add .option("spark.cosmos.preferredRegion"
3
4 df = spark.read
5 .format("cosmos.olap")
6 .option("spark.synapse.LinkedService", "AdventureworksMongoDB")
7 .option("spark.cosmos.container", "Sales")
8 .load()
9
10 display(df.limit(10))

Command executed in 1min 616ms by garyhop on 11-25-2020 01:15:29.762 -0800
> Job execution Succeeded Spark 2 executors 16 cores
View in monitoring Open Spark UI

View Table Chart
+---+---+---+---+---+---+---+
| _id | type | name | customerId | address | orderDate | shipDate | details |
+---+---+---+---+---+---+---+
| 156-0000... | "String" | "544887" | "String" | "customer" | "Franklin" | "2014-01-05T00:00:00Z" | "[{"String": "street", "String": "123 Main St", "String": "city", "String": "Anytown", "String": "state", "String": "CA", "String": "zip", "String": "95501"}]" |
| 156-0000... | "String" | "1000C39" | "String" | "salesOrder" | "2014-01-05T00:00:00Z" | "2014-02-05T00:00:00Z" | "[{"String": "details", "String": "Customer Order", "String": "status", "String": "Shipped", "String": "orderNumber", "String": "1000C39", "String": "orderDate", "String": "2014-01-05T00:00:00Z", "String": "shipDate", "String": "2014-02-05T00:00:00Z", "String": "customer", "String": "Franklin", "String": "customerID", "String": "544887"}]" |
+---+---+---+---+---+---+---+

```

6. Scrolling right though the result set

You will see that columns that contain JSON objects, such as address (8) and JSON arrays such a detail (9) has the JSON as their column content as well, however this too is expanded to include the data type information.

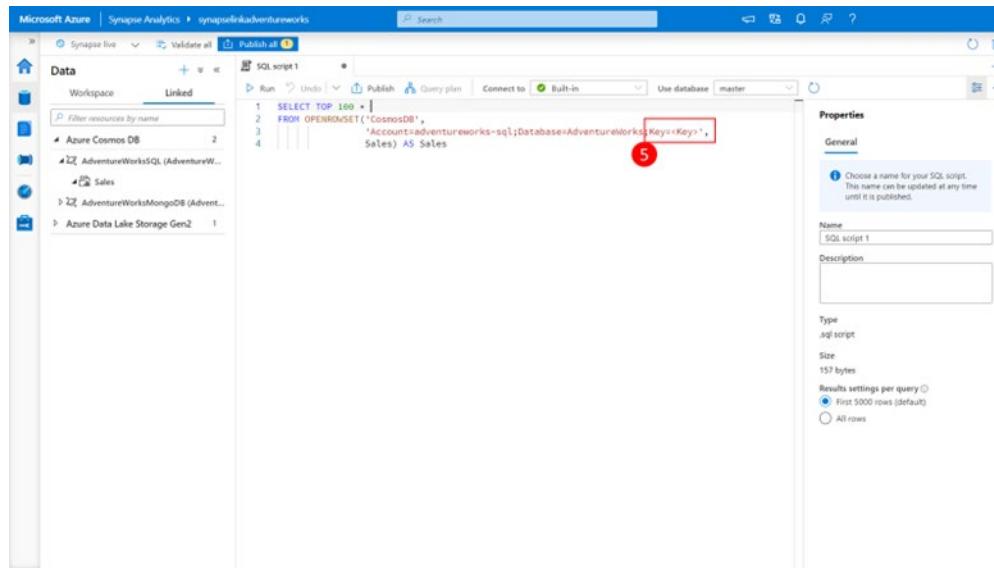
Validate connectivity from SQL Serverless

SQL Queries for Cosmos DB Core (SQL) API

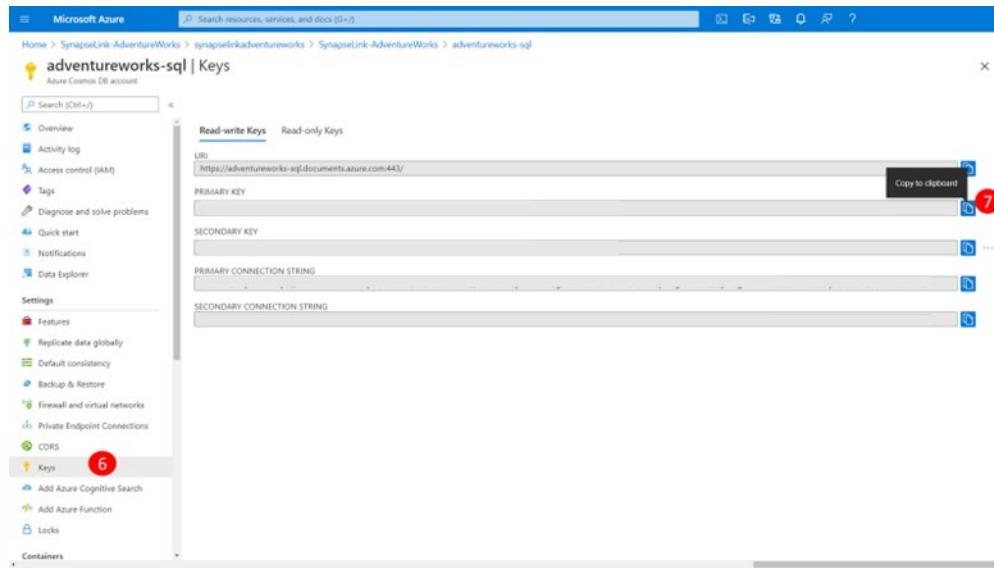
Let's now connect to our Cosmos DB Cosmos DB Core (SQL) API analytical store using the Synapse Analytics SQL Serverless capability and retrieve some data by performing the following steps:

1. Expand the **AdventureWorksSQL linked service** in the explorer view and click on the **Sale container** (1)

2. Click on the **Actions ellipsis “...”**
3. Click on **new notebook** to expose the list of New SQL Script actions **(3)**
4. Click on Select TOP 100 rows **(4)**, to load a SQL script window to retrieve the top 100 records from the Linked Server and its associated analytical store.



You will note that the SQL script template requires the Azure Cosmos DB account key **(5)** for the account we are trying to connect to.



5. You can retrieve this from the Azure Cosmos DB account by
 - a. Clicking on the **Keys** in the left-hand menu **(6)**
 - b. Clicking the **copy icon** next to the PRIMARY KEY value **(7)**

```

1 SELECT TOP 100 *
2 FROM OPENROWSET('CosmosDB',
3 'Account=adventureworks-sql;Database=AdventureWorks;Key=1K4aUTzg3Jl6x16rs2YMehr6vGnBpc7tgAII6VsafalYyElgn1nP17c1Lfcz2;
4 [Sales] AS Sales
    
```

6. And past the key value from the clipboard back into the query (5)

_id	_ts	_etag	Id	Type	Name	Customerid	OrderDate
mP0TAj5y9kC...	1606292304	'42001913-000...	000C23D8-BB8...	salesOrder	NULL	S4A887A7-8D89-4FAE-A668-AA9F43E26628	2014-02-16T00...
mP0TAj5y9kB...	16062915881	'42008809-000...	S4A887A7-8D8...	customer	Franklin Ye	S4A887A7-8D89-4FAE-A668-AA9F43E26628	NULL

7. Now we can click the **Run** button.

You should almost immediately see the query begin to execute and the shortly thereafter receive back a result set (7)

Note: That for records where data was not defined, such as the name column for the salesOrder record we get back a null value (8)

The screenshot shows the Microsoft Azure Synapse Analytics workspace. In the left sidebar, under the 'Data' section, there is a 'Linked' item which is expanded to show 'AdventureWorksMongoDB'. Below it, there is a 'Sales' container. The main area is a 'SQL script 1' window with the following script:

```
1 SELECT TOP 100 *
2 FROM OPENROWSET('CosmosDB',
3     'Account=adventureworks-mongodb;Database=Adventureworks;Key=VvJh4EE7qvDFr1CqL6ckYQQ0sdtsBEu8YpFchOKbQl7jDUVK48PKGY1';
4     Sales) AS Sales;
```

Below the script, the 'Results' tab is selected, showing the query results:

_id	_s	_etag	Id	Address	OrderDate	_id	Type	Name	Customer
1Fg1APSVy4B...	1606293078	'06009356-000...	NTR8Qgg3C...	{"object": "tree", ...}	NULL	["string"]"2014-01-01T00:00:00Z"	{"string"}"customer"	{"string"}"Franklin"	{"string"}"Retail"
1Fg1APSVy4B...	1606293181	'06009456-000...	MDAwGzrd0g...	NULL	["string"]"2014-01-01T00:00:00Z"	{"string"}"000C...	{"string"}"sales"	NULL	{"string"}"Retail"

A red box highlights the 'Address' column in the first row, and a red circle labeled 'A' is placed over the JSON object value.

You will see that columns that contain JSON objects, such as address **(A)** and JSON arrays such a detail has the JSON as their column value content.

8. You can now close the SQL script and discard the changes by clicking the Close and discard changes on the dialog that pops up

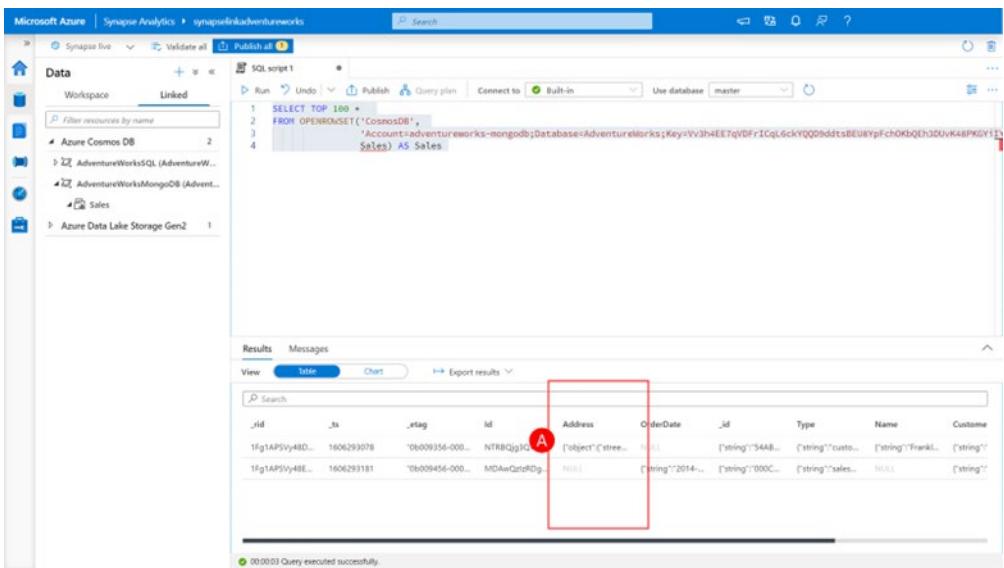
SQL Queries for Cosmos DB API for MongoDB

Let's now connect to our Cosmos DB Cosmos DB API for MongoDB analytical store using the Synapse Analytics SQL Serverless capability and retrieve some data by performing the following steps:

1. Expand the **AdventureWorksMongoDB linked service** in the explorer view and click on the **Sale container (1)**
2. Click on the **Actions ellipsis “...”**
3. Click on **new notebook** to expose the list of New SQL Script actions **(3)**
4. Click on **Select TOP 100 rows (4)**, to load a SQL script window to retrieve the top 100 records from the Linked Server and its associated analytical store.

You will see that the SQL script template requires the Azure Cosmos DB account key **(5)** for the account we are trying to connect to.

5. You can retrieve this from the Azure Cosmos DB account by
 - a. Click on the **Keys** in the left-hand menu
 - b. Click the **copy icon** next to the PRIMARY KEY value
6. And paste the key value from the clipboard back into the query
7. Now we can click the **Run** button.



A screenshot of the Microsoft Azure Synapse Analytics workspace. The left sidebar shows a 'Data' section with 'Workspace' and 'Linked' resources, including 'Azure Cosmos DB' (2), 'AdventureWorksSQL (AdventureW...', 'AdventureWorksMongoDB (Adventure...', and 'Sales'. The main area is titled 'SQL script 1' and contains the following T-SQL code:

```

1 SELECT TOP 100 *
2 FROM OPENROWSET('CosmosDB',
3     'Account=adventureworks-mongodb;Database=AdventureWorks;Key=VY3b4EE7qVDFr1CqL6ckvQ00ddtsBEUyPfchOKbQEh3DUvK48PKGY1',
4     '$.Sales') AS Sales;

```

The results pane shows a table with columns: _id, _st, _etag, id, Address, OrderDate, _id, Type, Name, and Customer. Two rows of data are visible. The second row, highlighted with a red box and labeled 'A', has an 'Address' column value of 'Object{"street": "123 Main St", "city": "Anytown", "state": "CA", "zip": "90210"}'.

You should almost immediately see the query begin to execute and the shortly thereafter receive back a result set.

Note: that for records where data was not defined, such as the name column for the salesOrder record we get back a null value and that the other column values are returned as JSON and expanded to include the properties data type.

Knowledge check

Question 1

Where do you enable Azure Synapse Link for Azure Cosmos DB?

- In Azure Synapse Analytics.
- In Azure Synapse Link.
- In Azure Cosmos DB.

Question 2

How do you disable Azure Synapse Link for Azure Cosmos DB?

- Delete the Azure Cosmos DB container.
- Delete the Azure Cosmos DB account.
- Set the Azure Synapse Link option to disable on the Azure Cosmos DB container.

Summary

In this module, you have learned how to leverage Azure Synapse Analytics Link for Cosmos DB to make operational data stored in Azure Cosmos DB available for analytical query from within Azure Synapse using both SQL and Spark.

In this module, you have:

- Enabled Cosmos DB Account to use Azure Synapse Link
- Created an analytical store enabled container
- Implemented Synapse Link for Cosmos DB
- Validated connectivity from Spark
- Validated connectivity from SQL Serverless

Query Azure Cosmos DB with Apache Spark for Azure Synapse Analytics

Introduction

In this module, you learn how to use the Synapse Spark to query the Azure Cosmos DB data made available by Azure Synapse Link.

After completing this module, you'll be able to:

- Query the Azure Cosmos DB analytical store
- Perform simple aggregations with analytical store data
- Perform cross container queries in Azure Cosmos DB
- Perform complex queries with JSON data
- Work with the windowing function and other aggregations
- Write data back to the Azure Cosmos DB transactional store
- Read data from the transactional store

Prerequisites

Before taking this module, it is recommended that the student is able to:

- Have a good understanding of Azure Cosmos DB capabilities
- Have a working knowledge of SQL or Spark
- Know how to deploy an Azure Synapse Analytics Workspace
- Know how to deploy an Azure Cosmos DB Account

Note: Throughout this module and those that follow we are going to be using two Azure Cosmos DB accounts, one configured for Azure Cosmos DB SQL (Core) API and a second account configured for the Azure Cosmos DB API for MongoDB in order to highlight the differences and similarities of the experience, specifically using their different default schema representation modes.

Query the Azure Cosmos DB analytical store

We are going to work with two new containers named Customer and SalesOrder. They contain Adventure Works datasets related to customer profile records and sales order records.

Each data set is stored in two different Azure Cosmos DB accounts. The customer profile data resides in a SQL API account. The sales order data resides in an Azure Cosmos DB API for MongoDB account.

Given that this data comes from separate systems, Adventure Works wants to use their available operational data to get insight into:

- What amount of revenue is coming from customers without completed profile data (no address details provided)
- How sales order volume and revenue are distributed by city for those customers where they do have address details.

To query the Azure Cosmos DB analytical store, perform the following steps:

1. Connect to an Azure Synapse Workspace that has an Azure Synapse SQL Serverless instance, and an Azure Synapse Spark Pool.

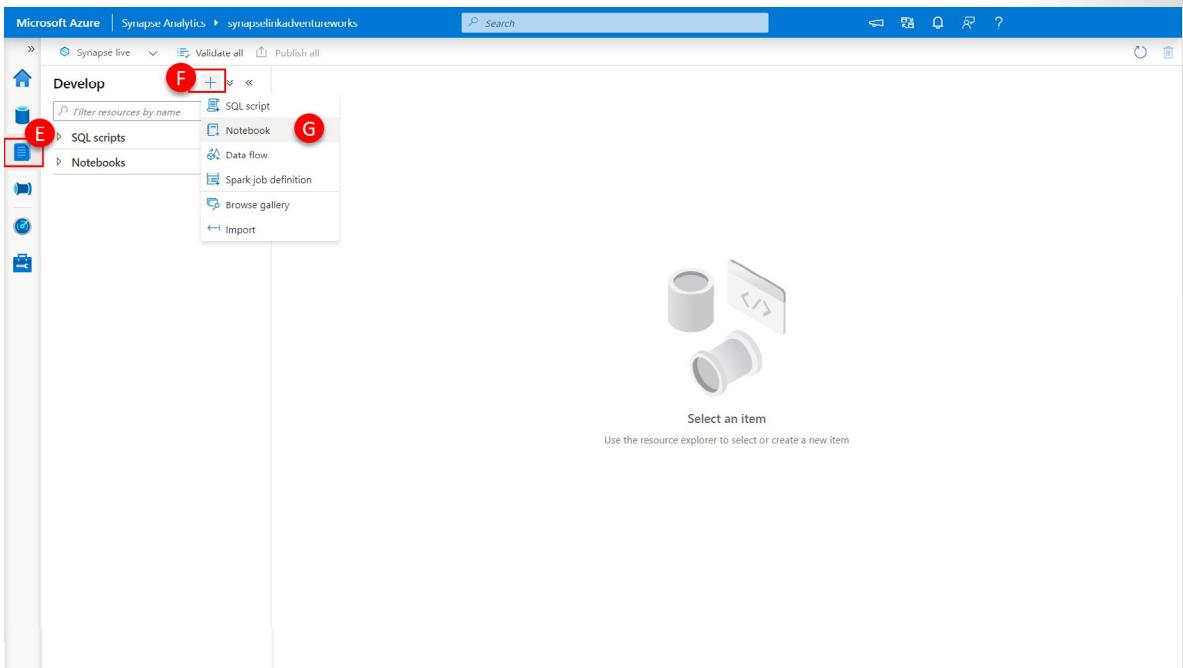
The screenshot shows the Microsoft Azure Synapse Analytics workspace interface. The left sidebar contains navigation links: Home, Data, Develop, Integrate, Monitor, and Manage. The main area is titled "Synapse workspace" and "synapselinkadventureworks". It features four cards: "Ingest" (Perform a one-time or scheduled data load), "Explore and analyze" (Learn how to get insights from your data), "Visualize" (Build interactive reports with Power BI capabilities), and "Learn" (Start with Azure Open Datasets and sample code). Below these are sections for "Recent resources" (empty) and "Feature showcase" (empty). A "Get started with Azure Synapse Analytics" button is visible.

2. In the left-hand menu, select **Data (A)**
3. Click on the **Linked tab** in the explorer view (B)
4. Expand the **AdventureWorksSQL** linked service to expose the **Customer** container
5. Expand the **AdventureWorksMongoDB** linked service to expose the **SalesOrder** container.

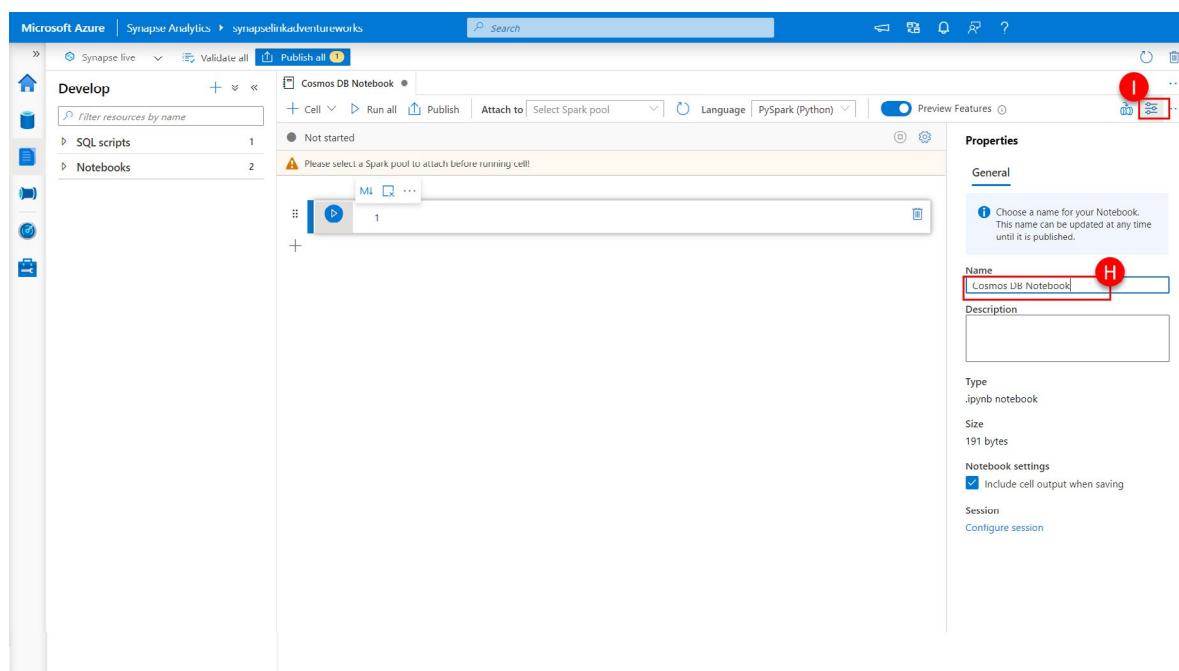
The screenshot shows the Azure Synapse Analytics Data Explorer. The left sidebar is labeled "Data" (with a red box A) and "Linked" (with a red box B). The main pane lists linked services: "Azure Cosmos DB" (2 items), "AdventureWorksSQL (AdventureWorks)" (Customer, Sales), "AdventureWorksMongoDB (AdventureWorks)" (Sales, SalesOrder), and "Azure Data Lake Storage Gen2" (1 item). Red arrows point from labels C and D to the "Customer" and "SalesOrder" containers respectively. A "Select an item" message at the bottom indicates "Use the resource explorer to select or create a new item".

Here you can see that an additional two containers are now visible in the data explorer view, under the previously created linked service to our Azure Cosmos DB accounts. The first, **Customer (C)**, has been created in the AdventureWorks database within the Azure Cosmos DB SQL API account and contains customer profile information. The second, **SalesOrder (D)**, has been created the Adventure-Works database within the Azure Cosmos DB API for MongoDB account and contains sales order information.

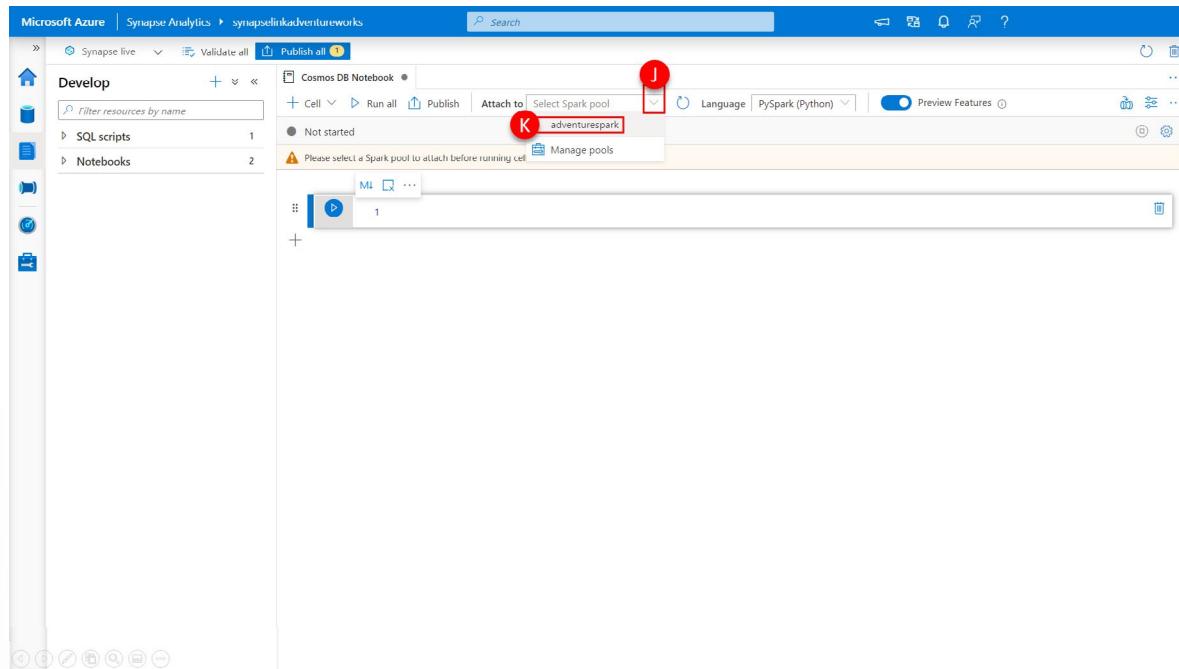
Let's open a new notebook to explore what is in these containers with Spark.



6. In the left-hand menu, select **develop (E)**
7. Then click the "+" (F) to add a resource.
8. And then select **Notebook (G)** to create a new notebook. A new notebook will immediately be created within the Synapse Workspace.



9. Within the notebook properties, provide an appropriate name, such as "**Cosmos DB Notebook**" (H).
10. And then click the **properties icon (I)** to close the properties.



Before we can run any spark jobs, we need to select the spark pool we wish to connect to execute our jobs.

11. Click the **attach to** dropdown at the top of the **notebook (J)** and select adventurespark, our previously deployed spark pool.

There is a language dropdown at the top of the notebook, this designates the default language for the notebook. We will be using both PySpark and Spark SQL, our default language will be PySpark and we will use the "%sql" magic to change to Spark SQL when appropriate.

Synapse Apache Spark allows you to analyze data in your Azure Cosmos DB containers that are enabled with Azure Synapse Link. The following two options are available to query the Azure Cosmos DB analytical store from Spark:

- Load to Spark DataFrame
- Create Spark table

Let's start by creating a DataFrame for each of our two containers and perform a read of the content of the analytical store.

```

dfCustomer = spark.read\
    .format("cosmos.olap")\
    .option("spark.synapse.linkedService", "AdventureWorksSQL")\
    .option("spark.cosmos.container", "Customer")\
    .load()

dfSalesOrder = spark.read\
    .format("cosmos.olap")\
    .option("spark.synapse.linkedService", "AdventureWorksMongoDB")\
    .option("spark.cosmos.container", "SalesOrder")\
    .load()

```

12. Click into the first cell of the **notebook (M)**

13. Paste the following python statements defining the two DataFrames we are going to use and the options for reading the data into them from the respective analytical stores:

```

dfCustomer = spark.read\
    .format("cosmos.olap")\
    .option("spark.synapse.linkedService", "AdventureWorksSQL")\
    .option("spark.cosmos.container", "Customer")\
    .load()

```

```

dfSalesOrder = spark.read\
    .format("cosmos.olap")\
    .option("spark.synapse.linkedService", "AdventureWorksMongoDB")\
    .option("spark.cosmos.container", "SalesOrder")\
    .load()

```

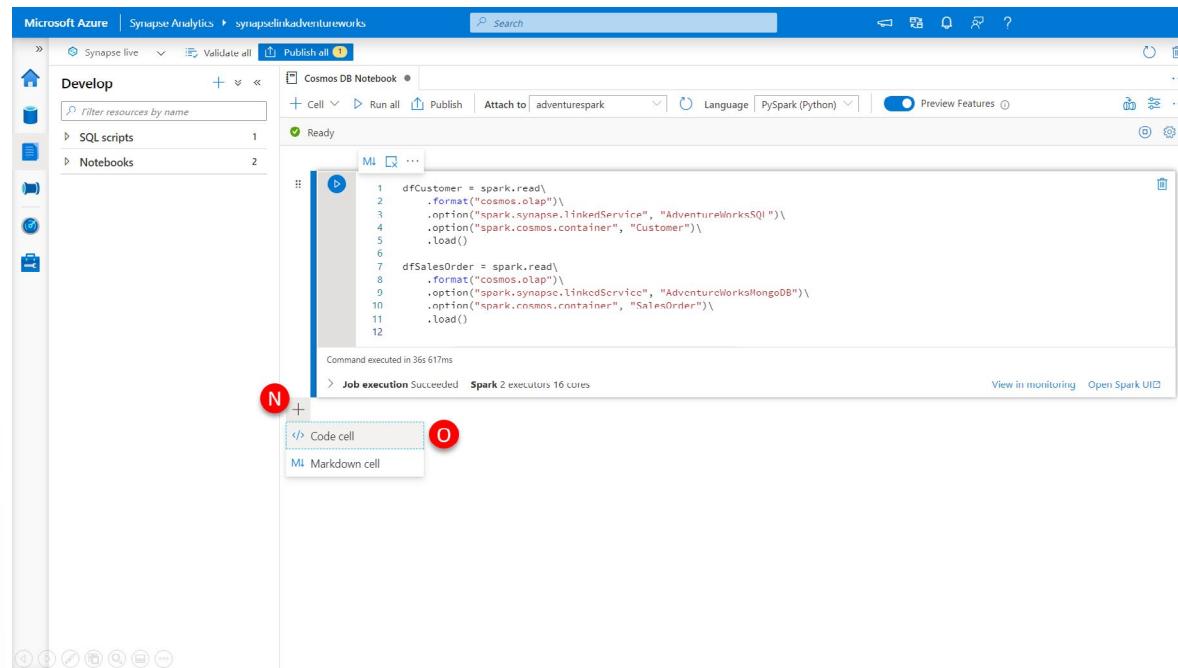
There are several parameters that need to be specified on the **spark.read** method to facilitate a read from Azure Cosmos DB analytical store:

- The format in the spark.read.format parameter needs to be specified as **cosmos.olap** to indicate that we are wanting to read from Azure Cosmos DB analytical store.
- An option should be set for **spark.synapse.linkedService** with the name of the previously create linked service.
- An option should be set for **spark.cosmos.container** specifying the name of the container that we wish to read.
- Optionally the **spark.cosmos.preferredRegions** option can be set to a list of preferred regions to use if you are using a Cosmos DB account with multiple regions configured.

14. Click the **run cell** button and within a couple of seconds the data from the respective analytical stores will be loaded into the two DataFrames we have defined.

Let's explore what data is contained in these DataFrames.

15. Click the "+" (N) at the bottom of notebook and select **Code Cell** to add an additional code cell.



The screenshot shows the Microsoft Azure Synapse Analytics workspace. In the center, there is a "Cosmos DB Notebook" titled "adventurespark". The notebook contains the following Python code:

```
1 dfCustomer = spark.read\
2     .format("cosmos.olap")\
3     .option("spark.synapse.linkedService", "AdventureworksSQL")\
4     .option("spark.cosmos.container", "Customer")\
5     .load()
6
7 dfSalesOrder = spark.read\
8     .format("cosmos.olap")\
9     .option("spark.synapse.linkedService", "AdventureworksMongoDB")\
10    .option("spark.cosmos.container", "SalesOrder")\
11    .load()
12
```

Below the code, a message indicates the command was executed successfully: "Command executed in 36s 617ms > Job execution Succeeded Spark 2 executors 16 cores". At the bottom of the notebook, there is a red circle labeled "N" next to a plus sign (+), which is used to add a new cell. A dropdown menu is open, showing "Code cell" (highlighted with a red circle labeled "O") and "Markdown cell".

16. Click into the new cell and paste the following.

```
display(dfCustomer.limit(10))
```

```
[2]
1 dfCustomer = spark.read\
2   .format("cosmos.olap")\
3   .option("spark.synapse.linkedService", "AdventureWorksSQL")\
4   .option("spark.cosmos.container", "Customer")\
5   .load()
6
7 dfSalesOrder = spark.read\
8   .format("cosmos.olap")\
9   .option("spark.synapse.linkedService", "AdventureWorksMongoDB")\
10  .option("spark.cosmos.container", "SalesOrder")\
11  .load()
12
```

Command executed in 366.617ms

Job execution Succeeded Spark 2 executors 16 cores

View in monitoring Open Spark UI

```
1 display(dfCustomer.limit(10)) P
```

17. Click the **run cell** button.

	title	firstName	lastName	emailAddress	phoneNumber	creationDate	address	password	eTag
i67-19A7-A292-F18CD7A7303C	Alejandro	Tang	alejandro30@adv...	1 (11) 500 555-0...	2013-10-11T00:0...	["addressLine1": "51 Jermyn Street", "city": "London", "state": "England", "zipCode": "SW1E 6BT", "country": "GB", "addressLine2": ""}]	["hash": "oXF5vQH"]	"56001d83-0000-0000-0000-000000000000"	
36E 44E4 BAAA 10F6344DD1A	Teresa	Blanco	teresa16@advent...	1 (11) 500 555 0...	2012-12-25T00:0...	["addressLine1": "123 Main Street", "city": "New York", "state": "NY", "zipCode": "10001", "country": "US", "addressLine2": ""}]	["hash": "gWPvJJK"]	"56002083 0000 0000 0000 000000000000"	
A9-4510-9BFB-73843C8709EF	Jonathan	Wright	jonathan51@adv...	164-555-0112	2014-03-03T00:0...	["addressLine1": "123 Main Street", "city": "New York", "state": "NY", "zipCode": "10001", "country": "US", "addressLine2": ""}]	["hash": "cNVy6fWz"]	"56002283-0000-0000-0000-000000000000"	
52-443b-8U/9-2bU/JUUhA8U	Meredith	Rodriguez	meredith19@adv...	1 (11) 500 555-0...	2014-02-06T00:0...	["addressLine1": "123 Main Street", "city": "New York", "state": "NY", "zipCode": "10001", "country": "US", "addressLine2": ""}]	["hash": "68JB5Gn"]	"56002483-0000-0000-0000-000000000000"	
4C-4929-AC27-F411ADC84729	Kyle	Campbell	kyle37@adventur...	372-555-0141	2014-01-18T00:0...	["addressLine1": "123 Main Street", "city": "New York", "state": "NY", "zipCode": "10001", "country": "US", "addressLine2": ""}]	["hash": "n1XgrzBF"]	"56002683-0000-0000-0000-000000000000"	
31-46D3-95D7-BAA31BA78F9	Dawn	Tang	dawn28@advent...	1 (11) 500 555-0...	2011-06-26T00:0...	["addressLine1": "123 Main Street", "city": "New York", "state": "NY", "zipCode": "10001", "country": "US", "addressLine2": ""}]	["hash": "j95pJfT"]	"56002883-0000-0000-0000-000000000000"	
78-4211-8336-88875B372E49	Ms.	Justine	justine0@advent...	498-555-0100	2013-12-07T00:0...	["addressLine1": "123 Main Street", "city": "New York", "state": "NY", "zipCode": "10001", "country": "US", "addressLine2": ""}]	["hash": "s8TQeLC"]	"56002a83-0000-0000-0000-000000000000"	
16-4E7A-88D2-6996B8A6F210	Dustin	Sharma	dustin9@advent...	1 (11) 500 555-0...	2011-11-07T00:0...	["addressLine1": "123 Main Street", "city": "New York", "state": "NY", "zipCode": "10001", "country": "US", "addressLine2": ""}]	["hash": "8QyobCU"]	"56002c83-0000-0000-0000-000000000000"	

You will be presented with a result set of the first 10 rows of a row-based representation of the documents contained within the Customer container's analytical store. The results are from an Azure Cosmos Core (SQL) API account, so that data will be represented using the well-defined schema representation by default.

You will note that the top-level properties of the document are represented as columns with the associated property values as the value of the column. In the case that these values are primitive data types ("string", "integer", "float" etc.) the column will be **typed (Q)**, if these properties are embedded

arrays or objects within the document, the column value will be a structure of these **embedded values (R)**.

In the case of our example, the title, firstName, lastName, emailAddress, and phoneNumber properties are primitive strings and assigned to their own **columns (Q)**, the address and password properties are **both embedded objects (R)**.

- Run a similar statement for the dfSalesOrder DataFrame, by creating a new cell, pasting the below and clicking the “run cell” button:

```
display(dfSalesOrder.limit(10))
```

_rid	_ts	id	_etag	_id	customerId	orderDate	shipDate	details
1Fg1ANPAqf0LA...	1607067351	RkNENENBNJMt...	"e10f231a-0000-...			"2013-07-16T08:55:56Z"	"2013-07-16T08:55:56Z"	<ul style="list-style-type: none"> ▼ ["string": "FC04CA", "string": "1658E5E", "string": "2013-07-16T08:55:56Z", "string": "FC04CA", "string": "1658E5E", "string": "2013-07-16T08:55:56Z", "string": "2013-07-16T08:55:56Z"]
1Fg1ANPAqf0LA...	1607067351	RkQwRTI0MUMt...	"e404251a-0000-...			"2013-08-26T00:00:00Z"	"2013-08-26T00:00:00Z"	<ul style="list-style-type: none"> ► ["string": "FD0E24", "string": "4669C9t", "string": "2013-08-26T00:00:00Z", "string": "FD0E24", "string": "4669C9t", "string": "2013-08-26T00:00:00Z", "string": "2013-09-01T00:00:00Z"]
1Fg1ANPAqf0LA...	1607067351	RkTDQkNCOEt...	"e4042a1a-0000-...			"2013-09-26T00:00:00Z"	"2013-09-26T00:00:00Z"	<ul style="list-style-type: none"> ► ["string": "FACBCBt", "string": "260D0D", "string": "2013-09-26T00:00:00Z", "string": "FACBCBt", "string": "260D0D", "string": "2013-09-26T00:00:00Z", "string": "2013-09-26T00:00:00Z"]
1Fg1ANPAqf0NA...	1607067351	RkixNTU3OTU0...	"e4042c1a-0000-...			"2014-03-28T00:00:00Z"	"2014-03-28T00:00:00Z"	<ul style="list-style-type: none"> ► ["string": "FB1557t", "string": "A8AF83t", "string": "2014-03-28T00:00:00Z", "string": "FB1557t", "string": "A8AF83t", "string": "2014-03-28T00:00:00Z", "string": "2014-03-28T00:00:00Z"]
1Fg1ANPAqf0RA...	1607067351	RjDNUT1OEetMT...	"e404301a-0000-...			"2013-08-11T00:00:00Z"	"2013-08-11T00:00:00Z"	<ul style="list-style-type: none"> ► ["string": "FHCSsxt", "string": "DC65t3", "string": "2013-08-11T00:00:00Z", "string": "FHCSsxt", "string": "DC65t3", "string": "2013-08-11T00:00:00Z", "string": "2013-08-11T00:00:00Z"]

You will now be presented with a result set of the first 10 rows of a row-based representation of the documents contained within the SalesOrder container’s analytical store. This is an Azure Cosmos Core API account for MongoDB, so that data will be represented using the full fidelity schema representation by default.

You will note that all top-level properties of the document are represented as columns with the associated property values as the value of the column. All properties are represented as a structure of the type of values assigned to the properties and the values themselves, (T) and (U). For complex types such as objects and arrays, these remain embedded within the structure but similarly expanded to include type encapsulation of each of their property values.

In this example the _id, customerId, orderDate, and shipDate are all strings and have a type of encapsulation of “string” (T). The details property is an embedded array (U), we can expand the structure within the column to reveal the three elements of the array [0],[1],[3] and in turn the embedded properties sku, name, price, and quantity of each array element object.

You will also note the presence of several Azure Cosmos DB system document properties (S). Azure Cosmos DB automatically has system properties such as _ts, _self, _attachments, _rid, and _etag associated with every document. These system document properties are seldom useful for analytical store query purposes and are easily removed by running the following PySpark code:

```

system_document_properties = {'_attachments','_etag','_rid','_self','_ts'}
customer_columns = list(set(dfCustomer.columns) - system_document_properties)
dfCustomer = dfCustomer.select(customer_columns)

display(dfCustomer.limit(10))

```

19. Paste the above code, click the **run cell** button.

This code defines the set of system property columns we wish to remove from the DataFrame, subtracts these from the list of all columns, and then selects just this subset of columns back into the DataFrame itself. If we display the resultant DataFrame, we see the **resultSet (R)** no longer contains these columns.

Similar code to remove system property columns from the dfSalesOrder DataFrame is as follows:

```

system_document_properties = {'_attachments','_etag','_rid','_self','_ts','id'}
salesorder_columns = list(set(dfSalesOrder.columns) - system_document_properties)
dfSalesOrder = dfSalesOrder.select(salesorder_columns)

```

```
display(dfSalesOrder.limit(10))
```

The screenshot shows the Microsoft Azure Synapse Analytics workspace. A job titled "Job execution Succeeded" has run successfully on "Spark 2 executors 16 cores". The code in the cell is:

```

1 # Remove unwanted system columns from the DataFrame
2 system_document_properties = {'_attachments','_etag','_rid','_self','_ts'}
3 customer_columns = list(set(dfCustomer.columns) - system_document_properties)
4 dfCustomer = dfCustomer.select(customer_columns)
5
6 display(dfCustomer.limit(10))

```

The resulting table view shows the following data:

firstName	password	address	id	creationDate	lastName	emailAddress	phoneNumber	title
Alejandro	► "[hash]":oXF5vQH	► "[addressLine1]":5	A8DBC223-4A67...	2013-10-11T00:0...	Tang	alejandro30@adv...	1 (11) 500 555-0...	
Teresa	► "[hash]":gWPvJ/Jk	► "[addressLine1]":7	A7DBD89C-496E...	2012-12-25T00:0...	Blanco	teresa16@advent...	1 (11) 500 555-0...	
Jonathan	► "[hash]":cNY6fjW,	► "[addressLine1]":5	A69F856B-0A9...	2014-03-03T00:0...	Wright	jonathan51@adv...	164-555-0112	
Meredith	► "[hash]":68IBSGM	► "[addressLine1]":5	A5930CBA-7952...	2014-02-06T00:0...	Rodriguez	meredith19@adv...	1 (11) 500 555-0...	
Kyle	► "[hash]":n1XgrZBf	► "[addressLine1]":6	A4B59163-EC4C...	2014-01-18T00:0...	Campbell	kyle37@adventur...	372-555-0141	
Dawn	► "[hash]":Jv95hp/T	► "[addressLine1]":6	A383596E-FC31...	2011-06-26T00:0...	Tang	dawn28@adventur...	1 (11) 500 555-0...	
Justine	► "[hash]":sE8TQeLk	► "[addressLine1]":2	A25RR3F4-7778...	2013-12-07T00:0...	Ryan	justine10@advent...	498-555-0100	
Dustin	► "[hash]":8QyabCL	► "[addressLine1]":4	A1156172-E416...	2011-11-07T00:0...	Sharma	dustin9@adventur...	1 (11) 500 555-0...	
Brittany	► "[hash]":4Xl0cfvc!	► "[addressLine1]":6	9F7FBEA7-AAB1...	2014-01-10T00:0...	Flores	brittany11@adve...	367-555-0114	
Raymond	► "[hash]":9+zECapi	► "[addressLine1]":6	9E410EFO-4850-4...	2013-08-23T00:0...	Martinez	raymond18@adve...	1 (11) 500 555-0...	

20. Paste the above code (X), click the **run cell** button.

```
1 # Remove unwanted system columns from the DataFrame
2 system_document_properties = {'_attachments','_etag','_rid','_self','_ts','_id'}
3 salesorder_columns = [x for x in dfSalesOrder.columns if x not in system_document_properties]
4 dfSalesOrder = dfSalesOrder.select(salesorder_columns)
5
6 display(dfSalesOrder.limit(10))
```

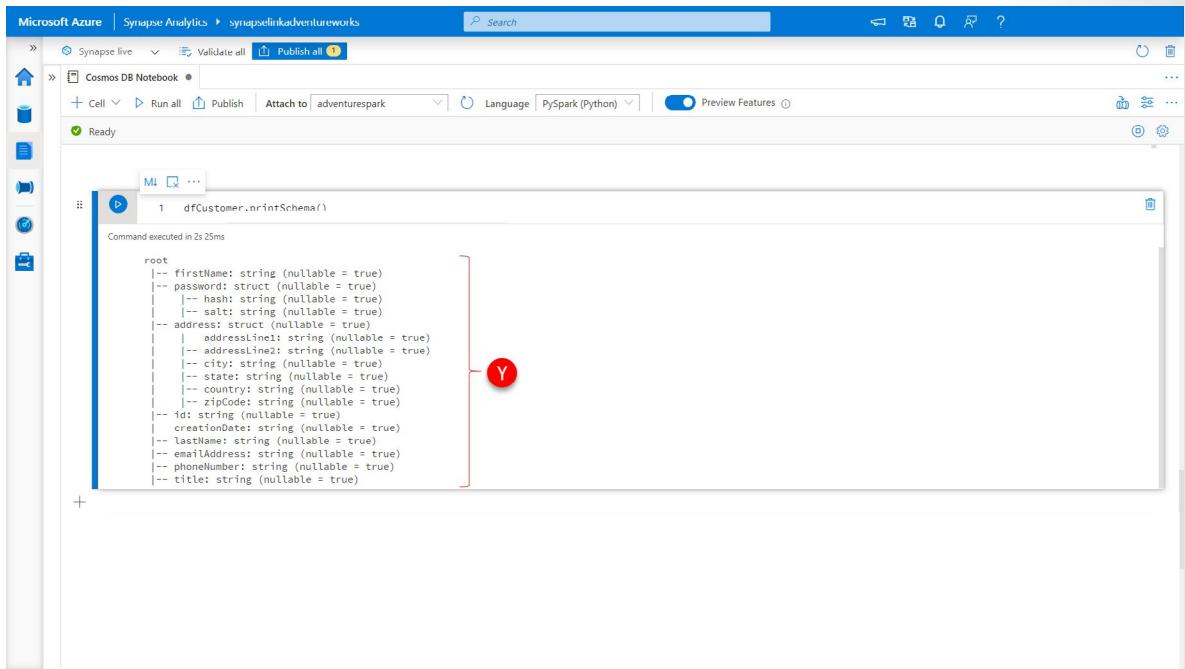
You will note that for this DataFrame we have additionally included the **ID column (W)**, for Azure Cosmos DB API for MongoDB accounts, the ID column can be considered a system document property and should not be confused with the `_id` property of the original document.

To get a more visual representation of the DataFrame schema, you can use the `printSchema()` method on the DataFrame.

21. Paste the code into a new cell (Y), and click the **run cell** button.

```
dfSalesOrder.printSchema()
```

It prints out the schema in an easy-to-read **tree format (Y)**, and you can easily see the structure of the address object within the `dfCustomer` DataFrame.



A screenshot of the Microsoft Azure Synapse Analytics workspace. The current notebook is titled "Cosmos DB Notebook". A single cell is active, containing the command `dfCustomer.printSchema()`. The output shows the schema of the DataFrame:

```

root
 |-- firstName: string (nullable = true)
 |-- password: string (nullable = true)
 |   |-- hash: string (nullable = true)
 |   |-- salt: string (nullable = true)
 |-- address: struct (nullable = true)
 |   |-- addressLine1: string (nullable = true)
 |   |-- addressLine2: string (nullable = true)
 |   |-- city: string (nullable = true)
 |   |-- state: string (nullable = true)
 |   |-- country: string (nullable = true)
 |   |-- zipCode: string (nullable = true)
 |-- id: string (nullable = true)
 |-- creationDate: string (nullable = true)
 |-- lastUpdate: string (nullable = true)
 |-- emailId: string (nullable = true)
 |-- phoneNumber: string (nullable = true)
 |-- title: string (nullable = true)

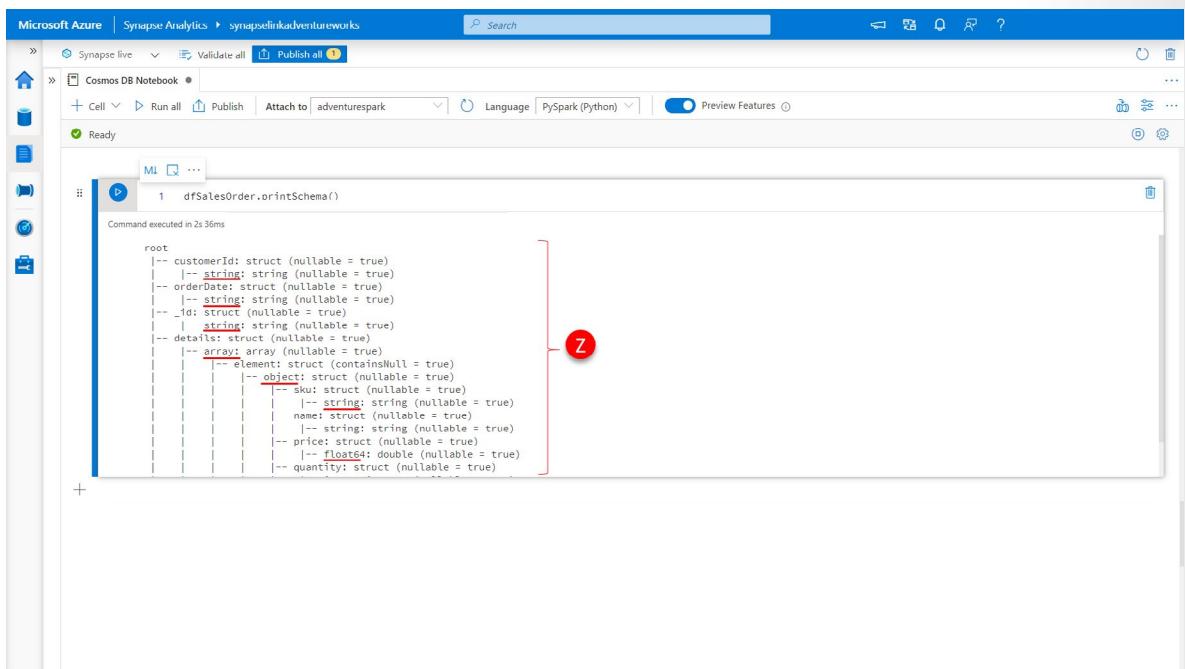
```

A red bracket labeled 'Y' highlights the entire schema definition.

And the following will do a similar thing for the dfSalesOrder DataFrame
`dfSalesOrder.printSchema()`

22. Paste the above code into a **new cell (Z)**, click the **run cell** button.

Here you can see the type of encapsulation (underlined) of each property value and the embedding of the details array within the SaleOrder document that contains the sales order line items with the sku, price, quantity etc. for each sales order, again encapsulated within the type of structure.



A screenshot of the Microsoft Azure Synapse Analytics workspace. The current notebook is titled "Cosmos DB Notebook". A new cell is active, containing the command `dfSalesOrder.printSchema()`. The output shows the schema of the DataFrame:

```

root
 |-- customerId: struct (nullable = true)
 |   |-- string: string (nullable = true)
 |-- orderId: struct (nullable = true)
 |   |-- string: string (nullable = true)
 |-- _id: struct (nullable = true)
 |   |-- string: string (nullable = true)
 |-- details: struct (nullable = true)
 |-- array: array (nullable = true)
 |   |-- element: struct (containsNull = true)
 |       |-- object: struct (nullable = true)
 |           |-- sku: struct (nullable = true)
 |               |-- string: string (nullable = true)
 |               |-- name: struct (nullable = true)
 |                   |-- string: string (nullable = true)
 |                   |-- price: struct (nullable = true)
 |                       |-- float4: double (nullable = true)
 |                   |-- quantity: struct (nullable = true)

```

A red bracket labeled 'Z' highlights the array element structure.

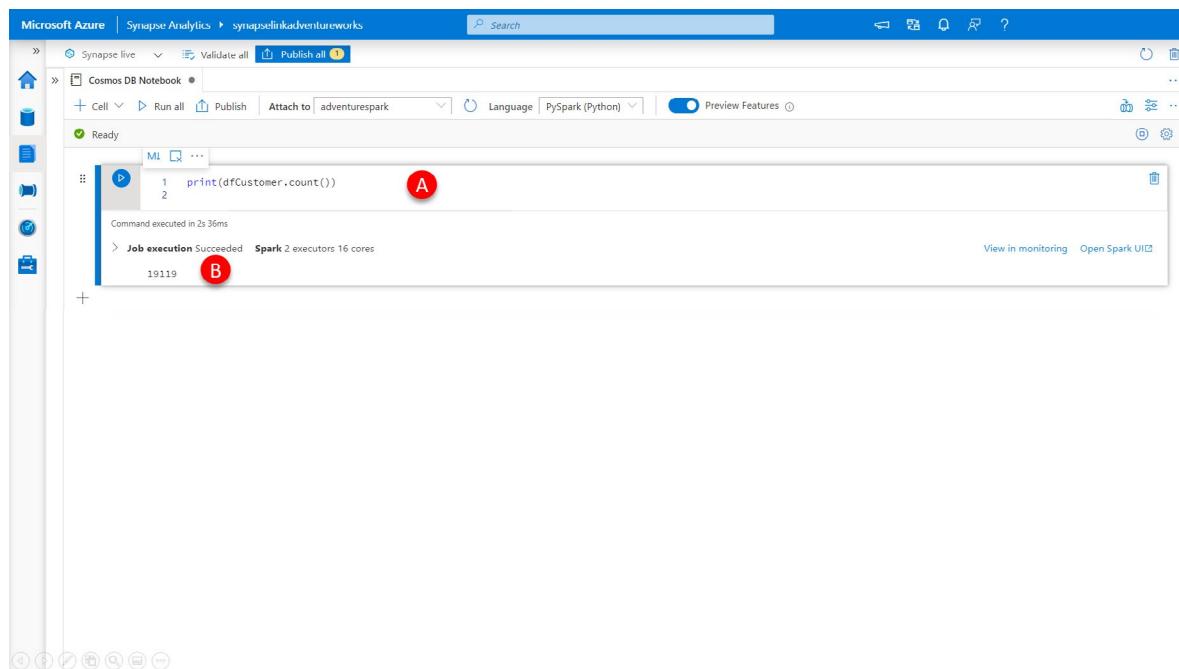
Perform simple aggregations with analytical store data

Now that we have explored the basic structure of the analytical store, lets dig a little deeper into what this data can tell us about the Adventure Works business.

Let's start by exploring some basic statistics, the simplest being the number of sales orders we have.

1. Paste the below code into a **new cell (A)**, click the **run cell** button.

```
print(dfCustomer.count())
```



You will see that there are **19119 customers (B)**

Let's break this down by country and city by customers, where we have the address information; and see how many customers there are where we don't have the address information captured on their customer profile.

2. Paste the below code into a **new cell (C)**, click the **run cell** button.

```
display(dfCustomer.groupBy("address.country","address.city").count().orderBy("count", ascending=False).limit(10))
```

You will see a breakdown of the top 10 country, city combinations having the most customers, right at the top of the list you will see that there are 615 customers for which Adventure Works has no country or city information within the customer profile.

The screenshot shows a Microsoft Azure Synapse Analytics notebook titled "Cosmos DB Notebook". A code cell displays the following PySpark command:

```
display(dfCustomer.groupBy("address.country","address.city").count().orderBy("count", ascending=False).limit(10))
```

The command was executed successfully in 2s 32ms using Spark 2 executors 16 cores. The output is displayed in a table view, showing the count of customers by country and city. A red circle labeled "D" points to the table output.

country	city	count
GB	London	420
FR	Paris	386
US	Burien	212
US	Concord	212
US	Beaverton	210
US	Bellingham	210
US	Chula Vista	206
US	Berkeley	200
US	Burlingame	198

Remember that we can always use the built-in chart capabilities of the Synapse Analytics notebooks to visualize our data more easily directly within the notebook.

The screenshot shows the same Microsoft Azure Synapse Analytics notebook. The chart properties panel (F) is open, and the chart type is set to "bar chart". The key is set to "country, city" and the values are set to "count". The aggregation is set to "SUM". A red circle labeled "E" points to the chart button in the toolbar.

The chart displays the count of customers by country and city. The y-axis represents the count, ranging from 0 to 700. The x-axis lists the country and city pairs. The highest bar corresponds to the entry "NULL" in the table, with a value of 635. A red circle labeled "F" points to the chart properties panel.

country	city	count
GB	London	420
FR	Paris	386
US	Beaverton	210
US	Bellingham	210
US	Chula Vista	206
US	Berkeley	200
US	Burlingame	198
NULL		635
US	Concord	212
US	Burien	212

3. Click the **Chart button (E)**
4. Set the **chart properties (F)**:
 - Chart type = bar graph
 - Key = country and city
 - Values = count

- Aggregation = SUM

As mentioned earlier, we are going to use both PySpark and Spark SQL, which we will include going forward. However, to be able to use both interchangeably within the same notebook, it is often useful to be able to work on the same data sets.

To save a PySpark DataFrame as a temporary view there is a DataFrame method.createOrReplaceView, which does just that. Temporary Views can also be queried from Spark SQL. Temporary views in Spark SQL are session scoped and will disappear if the session that creates it terminates.

To load data into a DataFrame using a Spark SQL query, you can use the **spark.sql()** method to run a query and return a DataFrame as a result.

Let's create a temporary view from our dfCustomer DataFrame and the query it back into a different DataFrame and display the result set.

5. Paste the code below into a new cell, click the **run cell** button, (G) and (H).

```
dfCustomer.createOrReplaceTempView("CustomerTempView")
```

```
dfResult = spark.sql("SELECT * FROM CustomerTempView")
display(dfResult.limit(10))
```

The screenshot shows the Microsoft Azure Synapse Analytics workspace interface. It features a top navigation bar with 'Microsoft Azure', 'Synapse Analytics', 'Cosmos DB Notebook', and various tool icons. Below the navigation is a toolbar with 'Cell', 'Run all', 'Publish', 'Attach to', 'Language' (set to PySpark (Python)), and 'Preview Features'. A status bar at the bottom indicates 'Ready'.

The main area contains two code cells:

- Cell 1 (Top):** Contains the Python code:

```
[9]: dfCustomer.createOrReplaceTempView("CustomerTempView")  
2  
3 dfResult = spark.sql("SELECT * FROM CustomerTempView")  
4 display(dfResult.limit(3))  
5
```

Red circle (G) points to the first line of code. Red circle (H) points to the closing brace of the code block.
- Cell 2 (Bottom):** Contains the SQL query:

```
1 %%sql  
2 SELECT * FROM CustomerTempView LIMIT 3
```

Red circle (I) points to the second line of code.

Below the code cells are the results of the execution:

- Cell 1 Result:** A table showing customer data with columns: firstName, password, liveData, address, id, creationDate, lastName, emailAddress, phoneNumber, title. The data for three rows is shown:

firstName	password	liveData	address	id	creationDate	lastName	emailAddress	phoneNumber	title
Alejandro	>{"hash": "oXF5vQH	>["addressLine1": "A8DBC223-4A07...	2013-10-11T00:0...	Teng	alejandro30@adv...	1 (11) 500 555-0...			
Teresa	>{"hash": "gWPvJlK	>["addressLine1": "ATDBD89C-496E...	2012-12-25T00:0...	Blanco	teresa16@advent...	1 (11) 500 555-0...			
Jonathan	> {"hash": "cNY6jW	> ["addressLine1": "A69F856B-0AA9...	2014-03-03T00:0...	Wright	jonathan51@adv...	164-555-0112			
- Cell 2 Result:** A table showing the schema of the temporary view:

firstName	password	liveData	address	id	creationDate
Alejandro	> [{"schema": [{"name": "hash", "dataT: null	> [{"schema": [{"name": "addressLine1": "A8DBC223-4A07-49A7-A292-F14...	2013-10-11T00:00:00		

You will see that the result set returned is the same as what was originally in our dfCustomer DataFrame and made the journey to temporary view and back unscathed.

If we query this same temporary view using Spark SQL by

6. Paste the below code into a new cell (I), click the "run cell" button.

```
%%sql
```

```
SELECT * FROM CustomerTempView LIMIT 10
```

You will again see the same result set, now delivered directly by running a Spark SQL query. We have specified the `%%sql` construct to inform the notebook that this cell contains Spark SQL code, not the default PySpark code it would otherwise be expecting.

Perform cross container queries in Azure Cosmos DB

Let's explore the Adventure Works data in more detail, and see what additional insights we can get by combining the data that is stored in the Azure Cosmos DB Core (SQL) API, and the Azure Cosmos DB API for MongoDB.

We are primarily going to use Spark SQL to explore this data, so most code will start with `%%sql` construct.

To begin, the two options available for querying the Azure Cosmos DB analytical store from Spark include:

- Loading to Spark DataFrame
- Creating Spark table

We have so far used loading the data into a DataFrame as the approach, lets create a Spark table to access our analytical store data by

The screenshot shows the Microsoft Azure Synapse Analytics workspace interface. On the left, the Data Explorer displays a workspace named 'synapseslinkadventureworks' containing a 'default (Spark)' database with 'Tables' named 'customers' and 'salesorders'. Two cells are visible in the notebook:

- Cell A:** Contains the following Spark SQL code:


```
1 %%sql
2 CREATE TABLE Customers USING cosmos.olap OPTIONS (
3   spark.synapse.linkedService 'AdventureWorksSQL',
4   spark.cosmos.container 'Customer'
5 )
```
- Cell B:** Contains the following Spark SQL code:


```
1 %%sql
2 CREATE TABLE SalesOrders USING cosmos.olap OPTIONS (
3   spark.synapse.linkedService 'AdventureWorksMongoDB',
4   spark.cosmos.container 'SalesOrder'
5 )
```

Both cells show a successful execution status with 'Job execution Succeeded' and 'Spark 2 executors 16 cores'. Red arrows from the 'customers' and 'salesorders' table names in the Data Explorer point to the respective cells in the notebook.

1. Paste the code below into a **new cell (A)**, click the **run cell** button.

```
%%sql
CREATE TABLE Customers USING cosmos.olap OPTIONS (
  spark.synapse.linkedService 'AdventureWorksSQL',
  spark.cosmos.container 'Customer'
)
```

2. Paste the code below into a **new cell (B)**, click the **run cell** button.

```
%%sql  
CREATE TABLE SalesOrders USING cosmos.olap OPTIONS (  
    spark.synapse.linkedService 'AdventureWorksMongoDB',  
    spark.cosmos.container 'SalesOrder'  
)
```

3. In the left-side menu, click **Data**.
4. Click **workspace**.
5. Expand the **Databases**, **default (Spark)** and **tables** container.

You now have two Spark tables customers and sales orders that are connected to the Azure Cosmos DB analytical stores in a manner like how we read data into our DataFrames.

The **CREATE TABLE** statement contains a **USING** clause that specifies the data source as **cosmos.olap**, specifying the Cosmos DB analytical store and has an **OPTIONS** clause that sets:

- **spark.synapse.linkedService** to the name of our previously create linked service.
- **spark.cosmos.container** specifying the name of the container.

You can also optionally set the **spark.cosmos.preferredRegions** option to a list of preferred regions to use if you are using a Cosmos DB account with multiple regions configured.

Additionally, you can override the default behavior of the table, which is to have a stable schema based on the analytical store schema as at the time of creation, by setting the **spark.cosmos.autoSchemaMerge** option to true. Set the **spark.cosmos.autoSchemaMerge** to true if you wish for the schema changes made to the Cosmos DB container and associated analytical store to be immediately reflected in the table.

So, lets query these tables now using Spark SQL.

6. Paste the below code into a **new cell (C)**, click the **run cell** button.

```
%%sql  
SELECT address.city AS City_Name, address.country AS Country_Name, count(*) as Address_Count  
    FROM Customers  
    GROUP BY address.city, address.country  
    ORDER BY Address_Count DESC  
    LIMIT 10
```

```

    %%sql
    SELECT address.city AS City_Name, address.country AS Country_Name, count(*) as Address_Count
    FROM Customers
    GROUP BY address.city, address.country
    ORDER BY Address_Count DESC
    LIMIT 10
  
```

City_Name	Country_Name	Address_Count
null	null	635
London	GB	420
Paris	FR	386
Concord	US	212
Burien	US	212
Bellingham	US	210
Revererton	US	210
Chula Vista	US	206
Berkeley	US	200
Burlingame	US	198

You will notice that the query is using the customer's table we have created to return a result set identical to the one we created using the PySpark example earlier.

Given the power of Spark SQL lets join the data from the Azure Cosmos DB Core (SQL) API, in the customers table, with the data that is contained in the Azure Cosmos DB API for MongoDB, in the sales order table.

- Paste the below code into a new cell, click the **run cell** button.

```

%%sql
CREATE OR REPLACE TEMPORARY VIEW SalesOrderView
AS
SELECT s._id.string as SalesOrderId,
       c.id AS CustomerId, c.address.country AS Country, c.address.city AS City,
       to_date(s.orderdate.string) AS OrderDate, to_date(s.shipdate.string) AS ShipDate
  FROM Customers c
 INNER JOIN SalesOrders s
    ON c.id = s.CustomerId.string
  
```

```

1 %sql
2 CREATE OR REPLACE TEMPORARY VIEW SalesOrderView
3 AS
4   SELECT s._id.string as SalesOrderId,
5         c._id AS CustomerId, c.address.country AS Country, c.address.city AS City,
6         to_date(s.orderdate.string) AS OrderDate, to_date(s.shipdate.string) AS ShipDate
7   FROM Customers c
8   INNER JOIN SalesOrders s
9     ON c._id = s.CustomerId.string
10

```

No Data Available!

[61] 1 %sql
2 SELECT * FROM SalesOrderView LIMIT 10

We're doing a lot here so let's break it down.

We are now using Spark SQL to create a temporary view called **SalesOrderView (D)**.

Within which we are **renaming the _id column from the SalesOrders table to SalesOrderId (F)**, as the _id property is the ID of all Adventure Works sales order records. We have also **accessed the string values for this column by specifying _id.string (E)**. In a similar manner, we are accessing the address.country and address.city properties embedded in the address of the Customer table.

We are converting the data type of the ship date and order date properties to the date data type using the Spark SQL to_date() function, remembering to access the string values using the shipdate.string and orderdate.string respectively from the **SalesOrder table (G)**.

And lastly, we are joining the Customers table with the SalesOrders table using the customer ID (in the case of Customers, this is stored in the column ID, and in the case of SalesOrder, the CustomerId column, remembering to access the string values using CustomerId.string).

As the Adventure Works sales order records are stored in Azure Cosmos DB API for MongoDB account, and these accounts use full fidelity schema representation by default, we need to include the type of suffix to access the property values for the SalesOrders table. The Adventure Works customer profile records are store in Azure Cosmos DB Core (SQL) API account and these accounts use well-defined schema representation, so we do not need to include the type suffix to access the property values.

If we want to see the result set from our newly created view, perform the following steps:

- Paste the below code into a **new cell (I)**, click the **run cell** button.

```
%sql
SELECT * FROM SalesOrderView LIMIT 10
```

We can now see we have a well-shaped sales order result set that includes the customer county and city values. We can use this to get some additional insights into the number of sales orders per country and city.

- Paste the below code into a **new cell (J)**, click the **run cell** button.

```
%%sql
SELECT Country, City, Count(*) AS Total_Orders
FROM SalesOrderView
GROUP BY Country, City
ORDER BY Total_Orders DESC
```

Country	City	Total_Orders
null	null	3806
GB	London	686
FR	Paris	522
CA	Cliffside	414
US	Burien	259
US	Concord	253
US	Chula Vista	252
DE	Berlin	251
US	Bellingham	251
US	Beaverton	246
CA	Shawnee	244

That's about as much insight as we can get from the sales order record without using the information contained in the details column and given that this data is an embedded array, we need to explore ways to surface this data in a more row-friendly format.

Perform complex queries with JSON data

We now need to unpack the sales order line-item data that is contained within an array embedded in the detail's column of our SalesOrders table to access the unit and revenue information that it contains.

Luckily, Spark SQL has two functions designed to assist us with this problem:

- `explode()`, which separates the elements of array into multiple rows and uses the default column name `col` for elements of the array.
- `posexplode()`, which separates the elements of array into multiple rows with positions and uses the column name `pos` for position, `col` for elements of the array.

Note: There are similar `explode()` and `posexplode()` functions in PySpark.

Firstly, lets remind ourselves what the data we are interested in looks like in the current SalesOrders table by

- Pasting the below code into a **new cell (A)**, click the **run cell** button.

```
%%sql
SELECT _id.string as SalesOrderId, details
FROM SalesOrders
```

LIMIT 10

The screenshot shows the Microsoft Azure Synapse Analytics workspace. In the center, there is a 'Cosmos DB Notebook' tab. Below it, a cell contains the following PySpark SQL code:

```
%sql  
SELECT _id.string as SalesOrderId, details  
FROM SalesOrders  
LIMIT 10
```

A red circle labeled 'A' is positioned over the 'Run cell' button in the toolbar above the notebook area. Another red circle labeled 'B' is positioned over the 'New cell' button in the same toolbar.

Here we can see our details column contains array data that needs to be expanded. Each element of the embedded array represents an individual line item of the sales order parent record the ID of which is contained in the `_id` column of the `SalesOrders` table.

If we expand this array using the `explode()` function, by:

2. Pasting the below code into a **new cell (B)**, click the **run cell** button.

```
%sql  
SELECT _id.string as SalesOrderId, explode(details.array)  
FROM SalesOrders  
LIMIT 10
```

The screenshot shows the Microsoft Azure Synapse Analytics workspace interface. The top navigation bar includes 'Microsoft Azure', 'Synapse Analytics', and 'synapsealinkadventureworks'. A search bar is at the top right. Below the header, there's a toolbar with icons for 'Synapse live', 'Validate dll', 'Publish all', 'Search', and other common operations.

The main area displays a 'Cosmos DB Notebook' titled 'adventurespark'. The notebook contains a single cell with the following PySpark SQL code:

```
1 %sql
2 SELECT _id.string as SalesOrderId, explode(details.array)
3   FROM SalesOrders
4 LIMIT 10
```

A red circle labeled 'B' highlights the number '1' in the first line of the code. Below the code, a message indicates the command was executed in 2s+43ms. The job execution status is shown as 'Succeeded' with 2 executors and 16 cores. There are links to 'View in monitoring' and 'Open Spark UI'.

The results section shows a table view of the data. The columns are 'SalesOrderid' and 'col 1'. The data consists of 10 rows of SalesOrderids, each followed by its corresponding exploded array schema. The table has a header row and 10 data rows.

We see that we get a new row for each element of the array, so each SalesOrderID in the SalesOrders table is now represented by one or more rows of line item detail, however we are unable to identify the line item row uniquely (without making some assumptions about other properties of the document, always dangerous).

So, let's try using the `poseexplode()` function, by performing the following step

3. Pasting the below code into a **new cell (C)**, click the **run cell** button.

```
%%sql
SELECT _id.string as SalesOrderId, poseplode(details.array)
    FROM SalesOrders
    LIMIT 10
```

```
%%sql
SELECT _id.string as SalesOrderId, poseplode(details.array)
FROM SalesOrders
LIMIT 10
```

Job execution Succeeded Spark 2 executors 16 cores

We see that we again get back a new row for each element of the array, so each SalesOrderId in the SalesOrders table is now represented by one or more rows of line item detail. However, the pos column now uniquely identifies the item detail and the order in which it appears in the array. We also note that the pos value is 0 based, and Adventure Works by convention numbers its line items starting at 1, so we will need to fix that as we use this data.

Let's create a SaleOrderDetailsView that encapsulates this sales order line item information for future use by:

4. Pasting the below code into a **new cell (D)**, click the **run cell** button.

```
%%sql
CREATE OR REPLACE TEMPORARY VIEW SalesOrderDetailsView
AS
SELECT Ax.SalesOrderId,
       pos+1 as SalesOrderLine,
       col.object.sku.string AS SKUCode,
       col.object.price.float64 AS Price,
       col.object.quantity.int32 AS Quantity
  FROM
  (
    SELECT _id.string as SalesOrderId, poseplode(details.array) FROM SalesOrders
  ) Ax
```

```
1 %sql
2 CREATE OR REPLACE TEMPORARY VIEW SalesOrderDetailsView
3 AS
4     SELECT Ax.SalesOrderId,
5            pos+1 as SalesOrderLine,
6            col.object.sku.string AS SKUCode,
7            col.object.price.float64 AS Price,
8            col.object.quantity.int32 AS Quantity
9    FROM
10    (
11        SELECT _id.string as SalesOrderId, poseplode(details.array) FROM SalesOrders
12    ) Ax
13
```

Command executed in 2s 29ms

No Data Available!

In this view, we have wrapped the inner subquery that uses the `poseplode()` function to extract the content of the array with some additional projection specifically projecting the `col.object.sku.string`, `col.object.price.float64` and `col.object.quantity.int32` values as `SKUCode`, `Price` and `Quantity` respectively, remembering to include their respective data type suffixes when accessing their values.

We have also incremented to the `pos` value by 1 before projecting it into the `SalesOrderLine` column of the result set to consider the Adventure Works uses order line numbers starting at 1 by convention.

If we want to see the result of this view, we can:

5. Paste the below code into a **new cell (E)**, click the **run cell** button.

```
SELECT *
FROM SalesOrderDetailsView
LIMIT 10
```

```
1 %%sql
2 SELECT *
3   FROM SalesOrderDetailsView
4 LIMIT 10
```

SalesorderId	SalesOrderLine	SKUCode	Price	Quantity
FCD4CA63-4B85-4FE1-AAD9-CE...	1	HL-U509-B	34.99	1
TCD4CA63-4B85-4FE1-AAD9-CE...	2	TI-M023	35	1
FCD4CA63-4B85-4FE1-AAD9-CE...	3	TT-M928	4.99	1
FD0E241C-9862-4E65-8BF0-89E6...	1	SH-W890-L	69.99	1
FD0E241C-9862-4E65-8BF0-89E6...	2	CA-1088	8.99	1
FACBCB8A-6F65-4750-BAE-DD...	1	SH-W890-S	69.99	1
FB155795-9C37-4EE1-8F3E-42C7...	1	BK-R79Y-48	1700.99	1
FB155795-9C37-4EE1-8F3E-42C7...	2	LJ-0192-L	49.99	1
FB95558A-1793-45CD-9DDD-CF...	1	TI-M823	35	1
FB95558A-1793-45CD-9DDD-CF...	2	TT-M928	4.99	1

And as you can see, we now get back a result set with each sales order line individually represented in a row with the SalesOrderId and SalesOrderLine uniquely identifying it along with the associated SKUCode, price, and quantity information.

If we want to validate the schema of the SalesOrderDetailsView, we can

6. Paste the below code into a **new cell (F)**, click the **run cell** button.

DESCRIBE SalesOrderDetailsView

```
1 %%sql
2 DESCRIBE SalesOrderDetailsView
```

col_name	data_type	comment
SalesorderId	string	null
SalesOrderLine	int	null
SKUCode	string	null
Price	double	null
Quantity	int	null

As you can see the data types for the SKUCode, Price and Quantity where automatically mapped back to the underlying data types contained within the array without the need to manually infer data types for these columns.

Now that we have our sales order details information in an easy-to-use format, we should easily be able to extract the insights Adventure Works set out to gather. This is the subject of the next unit.

Work with the windowing function and other aggregations

Adventure Works wants to be able to understand how the sales order volume and revenue is distributed by city for those customers where they have address details. In the previous units, we prepared a SalesOrderView that contains a row for every customer sales order with the country and city information for that customer where that information was available and a SalesOrderDetailsView that contains a row for every sale order line with information on the price and quantity associated with the product sold. Together these views contain all the raw data we need to answer these questions by:

1. Paste the code below into a **new cell (A)**, click the **run cell** button.

```
%%sql
SELECT o.Country, o.City,
       COUNT(DISTINCT o.CustomerId) Total_Customers,
       COUNT(DISTINCT d.SalesOrderId) Total_Orders,
       COUNT(d.SalesOrderId) Total_OrderLines,
       SUM(d.Quantity*d.Price) AS Total_Revenue,
       dense_rank() OVER (ORDER BY SUM(d.Quantity*d.Price) DESC) as Rank_Revenue,
       dense_rank() OVER (ORDER BY COUNT(DISTINCT d.SalesOrderId) DESC) as Rank_Orders,
       dense_rank() OVER (ORDER BY COUNT(d.SalesOrderId) DESC) as Rank_OrderLines,
       dense_rank() OVER (PARTITION BY o.Country ORDER BY SUM(d.Quantity*d.Price) DESC) as Rank_Revenue_Country
FROM SalesOrderView o
INNER JOIN SalesOrderDetailsView d
      ON o.SalesOrderId = d.SalesOrderId
WHERE Country IS NOT NULL OR City IS NOT NULL
GROUP BY o.Country, o.City
ORDER BY Total_Revenue DESC
LIMIT 10
```

The screenshot shows a Microsoft Azure Synapse Analytics workspace. In the top navigation bar, it says "Microsoft Azure | Synapse Analytics > synapseslinkadventureworks". Below the navigation bar is a toolbar with icons for "Synapse live", "Validate all", "Publish all", "Search", and other options. The main area is titled "Cosmos DB Notebook" and has a status bar indicating "Ready". A code editor window displays a SQL query:

```
1  %%sql
2  SELECT o.Country, o.City,
3      COUNT(DISTINCT o.CustomerId) AS Total_Customers,
4      COUNT(DISTINCT d.SalesOrderId) AS Total_Orders,
5      COUNT(d.SalesOrderId) AS Total_OrderLines,
6      SUM(d.Quantity*d.Price) AS Total_Revenue,
7      dense_rank() OVER (ORDER BY SUM(d.Quantity*d.Price) DESC) AS Rank_Revenue,
8      dense_rank() OVER (ORDER BY COUNT(DISTINCT d.SalesOrderId) DESC) AS Rank_Orders,
9      dense_rank() OVER (RANK BY COUNT(DISTINCT d.SalesOrderId) DESC) AS Rank_OrderLines,
10     dense_rank() OVER (PARTITION BY o.Country ORDER BY SUM(d.Quantity*d.Price) DESC) AS Rank_Revenue_Country
11  FROM SalesOrderView o
12  INNER JOIN SalesOrderDetailsView d
13  ON o.SalesOrderId = d.SalesOrderId
14  WHERE Country IS NOT NULL OR City IS NOT NULL
15  GROUP BY o.Country, o.City
16  ORDER BY Total_Revenue DESC
17  LIMIT 10
18
```

Below the code editor, a message states "Command executed in 10s 154ms". It also shows "Job execution Succeeded" with "Spark 2 executors 16 cores". At the bottom, there are "View" and "Table" buttons, and a "Chart" button. The "Table" button is selected. The resulting table has columns: Country, City, Total Customers, Total Orders, Total OrderLines, and Total Revenue. The data is as follows:

Country	City	Total Customers	Total Orders	Total OrderLines	Total Revenue
GB	London	420	686	1579	802810.2968999991
FR	Paris	386	522	1174	539725.7999999999
AU	Wollongong	105	202	411	330913.4665
AU	Warrnambool	105	203	416	327036.3681999997
AU	Bendigo	104	201	396	314568.7192999999
...

This query answers the questions being asked through traditional aggregation, as we are mostly interested in understanding the number (COUNT) or total (SUM) of values a GROUP BY clause that covers both **Country and City (B)** can answer most of the questions with **absolute values for the total number of customers, orders and order lines and the sum of revenue by City (C)**.

To answer the ranking part of the question, we use window functions. In essence a window function calculate a result for every row of a table based on a group of rows, called the frame. Every row can have a unique frame associated with it for that window function allowing you to concisely express and solve ranking, analytic, and aggregation problems in powerful yet simple a manner no other approach does.

Here we use the **dense_rank() function to calculate the rank of each city by revenue, number of orders and total order lines (D)**.

As well as the rank of each city within each country, by partitioning the dense_rank() window function by the country.

```

1  %%sql
2  SELECT o.Country, o.City,
3     COUNT(DISTINCT o.CustomerId) Total_Customers,
4     COUNT(DISTINCT d.SalesOrderId) Total_Orders,
5     COUNT(d.SalesOrderId) Total_OrderLines,
6     SUM(d.Quantity*d.Price) AS Total_Revenue,
7     dense_rank() OVER (ORDER BY SUM(d.Quantity*d.Price) DESC) as Rank_Revenue,
8     dense_rank() OVER (ORDER BY COUNT(DISTINCT d.SalesOrderId) DESC) as Rank_Orders,
9     dense_rank() OVER (ORDER BY COUNT(DISTINCT d.SalesOrderId) DESC) as Rank_OrderLines,
10    dense_rank() OVER (PARTITION BY o.Country ORDER BY SUM(d.Quantity*d.Price) DESC) as Rank_Revenue_Country
11   FROM SalesOrderView o
12   INNER JOIN SalesOrderDetailsView d
13     ON o.SalesOrderId = d.SalesOrderId
14   WHERE Country IS NOT NULL OR City IS NOT NULL
15   GROUP BY o.Country, o.City
16   ORDER BY Total_Revenue DESC
17   LIMIT 10
18

```

Command executed in 10s 154ms
Job execution Succeeded Spark 2 executors 16 cores

Total OrderLines	Total Revenue	Rank Revenue	Rank Orders	Rank OrderLines	Rank Revenue Country
1579	802810.2968999991	1	1	1	1
1174	539725.7999999999	2	2	2	1
411	330913.4665	3	23	20	1
416	327036.3681999997	4	22	27	2
396	314568.7192999999	5	24	33	3
...

Given the usefulness of these results we can encapsulate them in a temporary view for future use by:

- Paste the code below into a **new cell (E)**, click the **run cell** button.

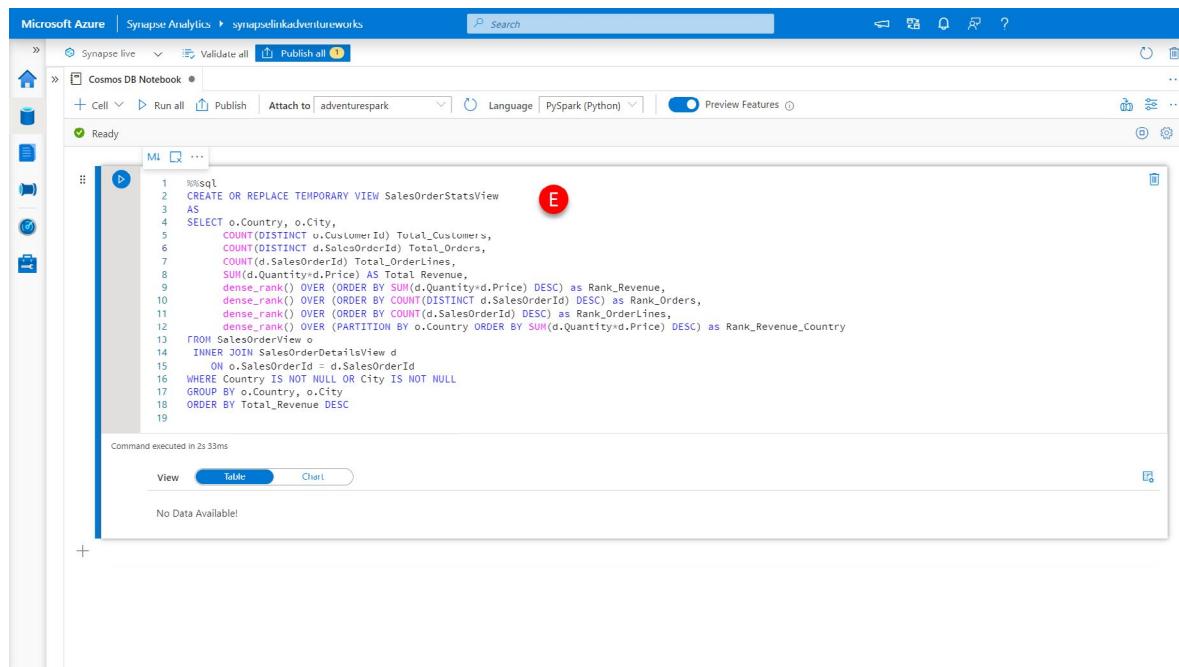
CREATE OR REPLACE TEMPORARY VIEW SalesOrderStatsView

AS

```

SELECT o.Country, o.City,
       COUNT(DISTINCT o.CustomerId) Total_Customers,
       COUNT(DISTINCT d.SalesOrderId) Total_Orders,
       COUNT(d.SalesOrderId) Total_OrderLines,
       SUM(d.Quantity*d.Price) AS Total_Revenue,
       dense_rank() OVER (ORDER BY SUM(d.Quantity*d.Price) DESC) as Rank_Revenue,
       dense_rank() OVER (ORDER BY COUNT(DISTINCT d.SalesOrderId) DESC) as Rank_Orders,
       dense_rank() OVER (ORDER BY COUNT(d.SalesOrderId) DESC) as Rank_OrderLines,
       dense_rank() OVER (PARTITION BY o.Country ORDER BY SUM(d.Quantity*d.Price) DESC) as Rank_Revenue_Country
  FROM SalesOrderView o
 INNER JOIN SalesOrderDetailsView d
   ON o.SalesOrderId = d.SalesOrderId
 WHERE Country IS NOT NULL OR City IS NOT NULL
 GROUP BY o.Country, o.City
 ORDER BY Total_Revenue DESC

```



```
%sql
CREATE OR REPLACE TEMPORARY VIEW SalesOrderStatsView
AS
SELECT o.Country, o.City,
       COUNT(DISTINCT o.CustomerId) Total_Customers,
       COUNT(DISTINCT d.SalesOrderId) Total_Orders,
       COUNT(d.SalesOrderId) Total_Orderlines,
       SUM(d.Quantity*d.Price) AS Total_Revenue,
       dense_rank() OVER (ORDER BY SUM(d.Quantity*d.Price) DESC) as Rank_Revenue,
       dense_rank() OVER (ORDER BY COUNT(DISTINCT d.SalesOrderId) DESC) as Rank_Orders,
       dense_rank() OVER (ORDER BY COUNT(d.SalesOrderId) DESC) as Rank_OrderLines,
       dense_rank() OVER (PARTITION BY o.Country ORDER BY SUM(d.Quantity*d.Price) DESC) as Rank_Revenue_Country
FROM SalesOrderView o
INNER JOIN SalesOrderDetailsView d
ON o.SalesOrderId = d.SalesOrderId
WHERE Country IS NOT NULL OR City IS NOT NULL
GROUP BY o.Country, o.City
ORDER BY Total_Revenue DESC
```

Write data back to the Azure Cosmos DB transactional store

Whilst the analytical insights that Adventure Works was able to gather from joining together operational data that was previously siloed in disconnected and hard to correlate data stores was useful, they have decided that making this information to every user across the organization through their existing application is the right way to achieve ongoing visibility of these key statistics. To do that we need to write this data back to the Azure Cosmos DB transactional stores.

As with all applications the process starts by appropriately modeling the data. Some of the things to consider as we write this data back to Azure Cosmos DB:

- Does the document have an appropriate partition key, such that it will evenly distribute items and queries across all partitions as the data grows.
- Can we provide an appropriate and usable primary key (ID or _id, in the case of Core (SQL) API and API for MongoDB respectively) such that when used together with the partition key can uniquely identify an item for point operations (CRUD)
- Where we have modeled containers to support multiple entity types that we identify the new data with the appropriate entity type.

For our example, where we are wanting to write the limited number of statistic rows back to the transactional store, we choose to create a new compound ID value that will allow client applications to look up the latest statistics for their city easily by concatenating the country code and city name together to uniquely identify each document. Given that we will be storing this in a container with other entity types we uniquely identify it type as "SalesOrderStatistic" data.

1. Paste the code below into a **new cell (A)**, click the **run cell** button.

```
%%sql
SELECT concat(Country,'-',replace(City,' ','')) AS id,
      'SalesOrderStatistic' AS type, *
```

```
FROM SalesOrderStatsView
LIMIT 10
```

The screenshot shows the Microsoft Azure Synapse Analytics workspace. In the center, there's a 'Cosmos DB Notebook' tab. A red circle labeled 'A' is over the first cell, which contains the following SQL code:

```
1 %%sql
2 SELECT concat(Country,'-',replace(City,' ','')) AS id,
3       'SalesOrderStatistic' AS type,
4       * FROM SalesOrderStatsView
5 LIMIT 10
```

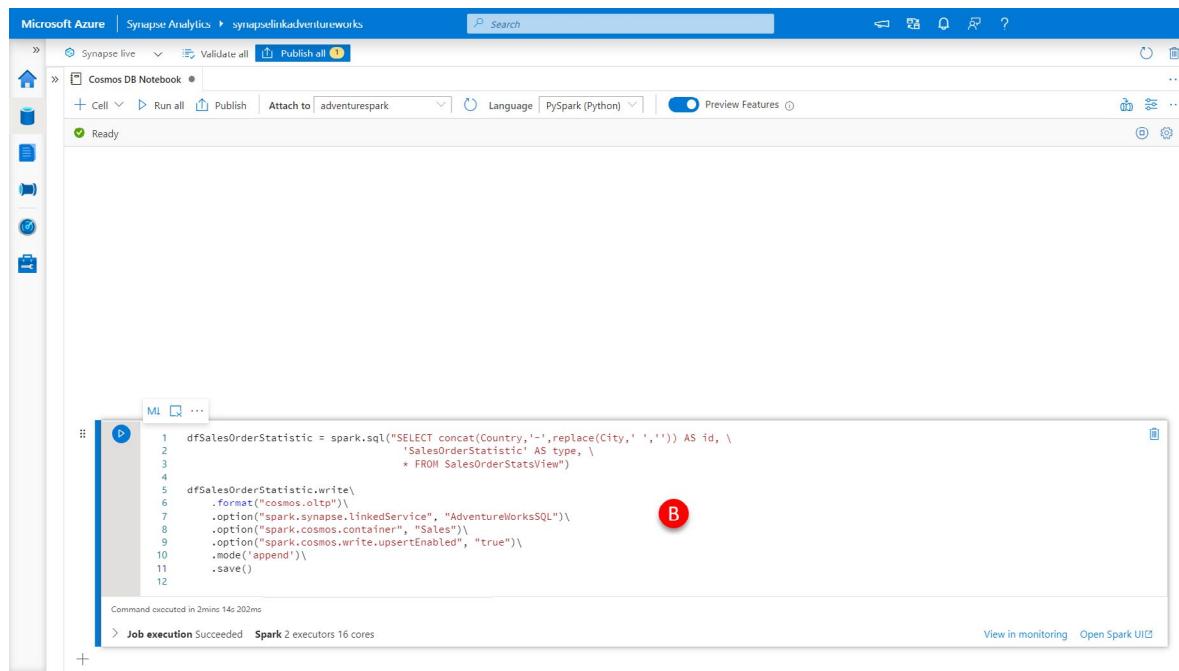
Below the code, it says "Command executed in 2s 50ms". Underneath, it shows "Job execution Succeeded" and "Spark 2 executors 16 cores". There are tabs for "View", "Table" (which is selected), and "Chart". The resulting data is displayed as a table:

id	type	Country	City	Total_Customers	Total_Orders
GB-London	SalesOrderStatistic	GB	London	420	686
FR-Paris	SalesOrderStatistic	FR	Paris	386	522
AU-Wollongong	SalesOrderStatistic	AU	Wollongong	105	202
AU-Warrnambool	SalesOrderStatistic	AU	Warrnambool	105	203
AU-Bendigo	SalesOrderStatistic	AU	Bendigo	104	201
AU-Goulburn	SalesOrderStatistic	AU	Goulburn	106	200
US-Bellflower	SalesOrderStatistic	US	Bellflower	194	243
AU-Brisbane	SalesOrderStatistic	AU	Brisbane	106	191
AU-Townsville	SalesOrderStatistic	AU	Townsville	105	199
AU-Geelong	SalesOrderStatistic	AU	Geelong	106	206

As you can see, we now have an appropriately shaped result set ready to write back to the transactional store, by

- Paste the code below into a **new cell (B)**, click the **run cell** button.

```
dfSalesOrderStatistic = spark.sql("SELECT concat(Country,'-',replace(City,' ','')) AS id,\n                                  'SalesOrderStatistic' AS type,\n                                  * FROM SalesOrderStatsView")\n\ndfSalesOrderStatistic.write\\n    .format("cosmos.oltp")\\n    .option("spark.synapse.linkedService", "AdventureWorksSQL")\\n    .option("spark.cosmos.container", "Sales")\\n    .option("spark.cosmos.write.upsertEnabled", "true")\\n    .mode('append')\\n    .save()
```



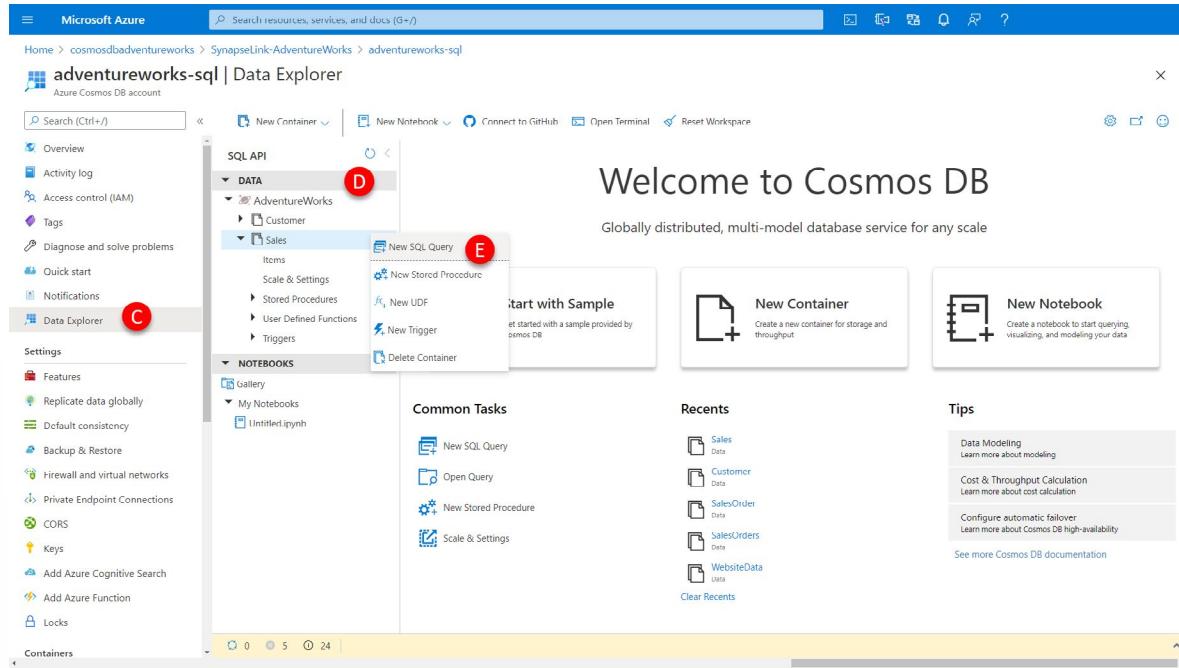
```

1 dfSalesOrderStatistic = spark.sql("SELECT concat(Country,'-',replace(City,' ','')) AS id, \
2                                     'SalesOrderStatistic' AS type, \
3                                     * FROM SalesOrderStatsView")
4
5 dfSalesOrderStatistic.write\
6   .format("cosmos.oltp")\
7   .option("spark.synapse.linkedService", "AdventureWorksSQL")\
8   .option("spark.cosmos.container", "Sales")\
9   .option("spark.cosmos.write.upsertEnabled", "true")\
10  .mode("append")\
11  .save()
12

```

Command executed in 2mins 14s 202ms
> Job execution Succeeded Spark 2 executors 16 cores

Within a couple of minutes the DataFrame write operation should have finished writing our statistics back to the transactional store.



Welcome to Cosmos DB

Globally distributed, multi-model database service for any scale

New Container New Notebook

Start with Sample

New SQL Query New StoredProcedure

New UDF New Trigger

New Container Create a new container for storage and throughput

New Notebook Create a notebook to start querying, visualizing, and modeling your data

Common Tasks

- New SQL Query
- Open Query
- New StoredProcedure
- Scale & Settings

Recents

- Sales Data
- Customer Data
- SalesOrder Data
- SalesOrders Data
- WebsiteData Data

Tips

- Data Modeling
- Cost & Throughput Calculation
- Configure automatic failover

See more Cosmos DB documentation

Let's validate that these records are now visible in our Cosmos DB container by.

3. Navigate to the Azure portal (<https://portal.azure.com>) and select the **Azure Cosmos DB account**.
4. Navigate to your previously created **Azure Cosmos DB Core (SQL) API account**.
5. Select **Data Explorer** in the **left-hand menu (C)**.

6. Navigate to the **Sales container** we created earlier by (D):
 - Expanding AdventureWorks database
 - Expanding the Sales Container
7. Clicking on the ... and selecting **New SQL Query** from the menu (E)
8. Paste the below code into the query pane (F), click the **Execute** button (G).

```
SELECT * FROM c WHERE c.type = "SalesOrderStatistic"
```

This will immediately return items containing the statistics we inserted from Synapse Analytics.

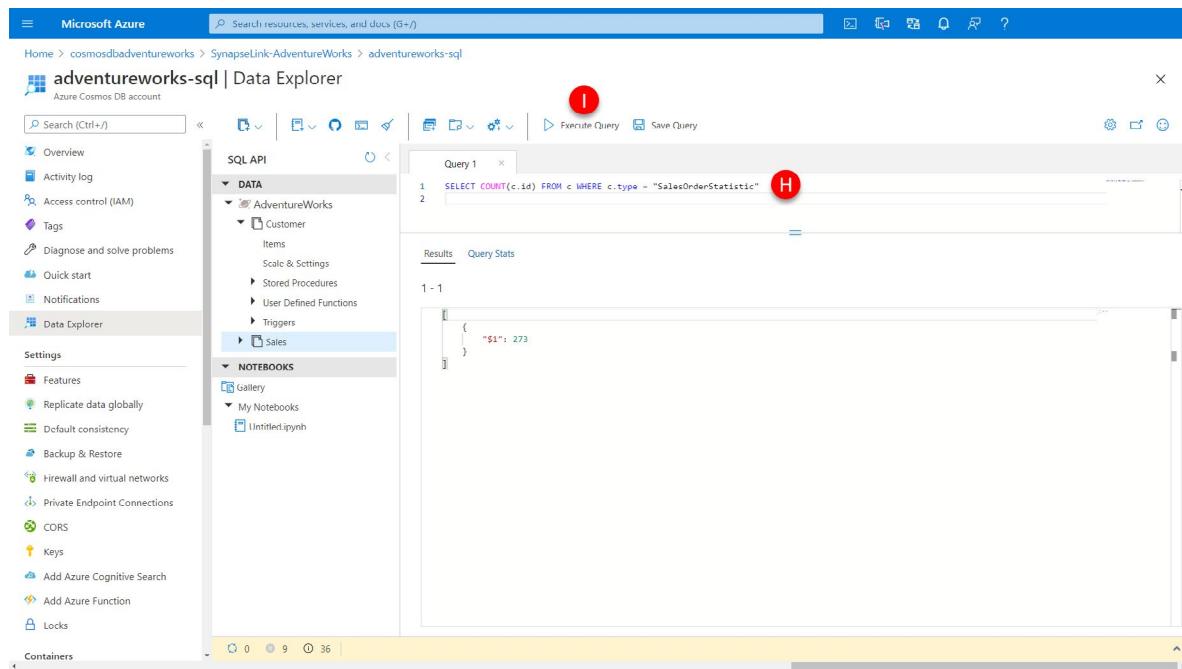
The screenshot shows the Microsoft Azure Data Explorer interface. The left sidebar shows the Azure account structure, including the 'adventureworks-sql' database and its 'Sales' container. The main pane displays a query results table with two columns: 'Rank_Orders' and 'Rank_Revenue'. The table contains 273 rows of data, each representing a SalesOrderStatistic item with various properties like Country, ID, and Revenue.

Rank_Orders	Rank_Revenue
27	23
35	23
AU	AU
AU-Hubert	AU-Hubert
"SalesOrderStatistic"	"SalesOrderStatistic"
197	197
186	186
239937.9633	239937.9633
17	17
"P07AJSjy9kfAAAAAA=--"	"P07AJSjy9kfAAAAAA=--"
"self": "/dbss/P07AJSjy9kfAAAAAA=--/colls/mP07AJSjy9k-/docs/mP07AJSjy9kfAAAAAA=--/"	"self": "/dbss/P07AJSjy9kfAAAAAA=--/colls/mP07AJSjy9k-/docs/mP07AJSjy9kfAAAAAA=--/"
"etag": "\\"3701fa5-0000-0000-0000-5fcbe468000\\\"	"etag": "\\"3701fa5-0000-0000-0000-5fcbe468000\\\"
"attachments": "attachments/",	"attachments": "attachments/",
"_ts": 1007147078	"_ts": 1007147078

We can validate that all the expected 273 items have been inserted by

9. Paste the code below into the query pane (F), click the **Execute** button (G).

```
SELECT COUNT(c.id) FROM c WHERE c.type = 'SalesOrderStatistic'
```



And should immediacy return a value of 273 as the result.

We have successfully updated the Azure Cosmos DB transactional store from Synapse Analytics with the results of our analytical processing done over data that was sourced from Azure Cosmos DB analytical store.

Read data from the transactional store

There may be scenarios where you may need to read data from the Azure Cosmos DB transactional store, for example, in where you cannot tolerate the near-real-time latency constraints of the analytical store to surface specific information or where schema inference rules have made data unavailable in the analytical store.

When connecting to the transactional store any queries, you run will consume Azure Cosmos DB request units and could impact workloads running against the transactional store. For most analytical type queries from Spark the analytical store is going to be your best choice from both a performance and cost perspective.

Now that you are familiar with reading from analytical store into a DataFrame reading from the transactional store will be simple as it requires only a single change to the query. For the transactional store, the format parameter is specified as **cosmos.oltp** instead of the **cosmos.olap** used to access the analytical store.

1. Paste the code below into a **new cell (A)**, click the **run cell** button.

```
dfCustomer = spark.read\  
.format("cosmos.oltp")\  
.option("spark.synapse.linkedService", "AdventureWorksSQL")\  
.option("spark.cosmos.container", "Customer")\  
.load()  
  
display(dfCustomer.limit(10))
```

The screenshot shows a Microsoft Azure Synapse Analytics workspace. In the center, there's a "Cosmos DB Notebook" tab. Below it, a code cell contains the following PySpark code:

```

1 dfCustomer = spark.read\
2   .format("cosmos.oltp")\
3   .option("spark.synapse.linkedService", "AdventureWorksSQL")\
4   .option("spark.cosmos.container", "Customer")\
5   .load()
6
7 display(dfCustomer.limit(10))
  
```

Below the code cell, a message says "Command executed in 4s 73ms". Underneath, it shows "Job execution Succeeded" and "Spark 2 executors: 16 cores". To the right, there are links to "View in monitoring" and "Open Spark UI".

At the bottom, there's a table view with the following columns: _attachments, _etag, _rid, _self, _ts, address, creationDate, emailAddress, firstName, and id. The table displays 10 rows of data from the Azure Cosmos DB transactional store.

_attachments	_etag	_rid	_self	_ts	address	creationDate	emailAddress	firstName	id
attachments/	"56000a83-0000-...	mP07Aluc-AgBA...	db5/mP07AA==/...	1607057561	>["addressLine1": "1	2011-09-07T00:00:00.000Z	samantha6@adventureworks.com	Samantha	A8D0C3BB-50CC-40E0-8000-000000000000
attachments/	"56000b83-0000-...	mP07Aluc-AgCA...	db5/mP07AA=-/...	1607057561	>["addressLine1": "2	2013-01-06T00:00:00.000Z	samuel21@adventureworks.com	Samuel	A7C84443-A5AD-4000-B000-000000000000
attachments/	"56000c83-0000-...	mP07Aluc-AqDA...	db5/mP07AA==/...	1607057561	>["addressLine1": "3	2014-01-19T00:00:00.000Z	edward67@adventureworks.com	Edward	A695E5F8-DE3A-4000-B000-000000000000
attachments/	"56000d83-0000-...	mP07Aluc-AgEA...	db5/mP07AA==/...	1607057561	>["addressLine1": "4	2014-01-23T00:00:00.000Z	henry4@adventureworks.com	Henry	A59053FD-3D9A-4000-B000-000000000000
attachments/	"56000e83-0000-...	mP07Aluc-AgFA...	db5/mP07AA=-/...	1607057561	>["addressLine1": "5	2013-11-22T00:00:00.000Z	jesse43@adventureworks.com	Jesse	A4B1FE7A-28AD-4000-B000-000000000000
attachments/	"56000f83-0000-...	mP07Aluc-AgGA...	db5/mP07AA==/...	1607057561	>["addressLine1": "6	2012-02-10T00:00:00.000Z	robert42@adventureworks.com	Robert	A3803AB7-9527-4000-B000-000000000000
attachments/	"56001083-0000-...	mP07Aluc-AgHA...	db5/mP07AA==/...	1607057561	>["addressLine1": "7	2013-10-23T00:00:00.000Z	jillian5@adventureworks.com	Jillian	A236A6BC-4E02-4000-B000-000000000000
attachments/	"56001183-0000-...	mP07Aluc-AgIA...	db5/mP07AA==/...	1607057561	>["addressLine1": "8	2014-05-03T00:00:00.000Z	tina11@adventureworks.com	Tina	A102A6F9-9588-4000-B000-000000000000
attachments/	"56001283-0000-...	mP07Aluc-AgJA...	db5/mP07AA=-/...	1607057561	>["addressLine1": "9	2013-07-11T00:00:00.000Z	kari7@adventureworks.com	Kari	9F7FA5B2-54FB-4000-B000-000000000000
attachments/	"56001383-0000-...	mP07Aluc-AgKA...	db5/mP07AA==/...	1607057561	>["addressLine1": "10	2013-11-24T00:00:00.000Z	dana17@adventureworks.com	Dana	9E35C07B-DB41-4000-B000-000000000000

And a similar approach applies to the creating a Spark SQL table to access the transactional store, the USING clause is changed to specify cosmos.oltp as the source.

- Paste the code below into a **new cell (B)**, click the **run cell** button.

```

%%sql
CREATE TABLE CustomersOLTP USING cosmos.oltp OPTIONS (
    spark.synapse.linkedService 'AdventureWorksSQL',
    spark.cosmos.container 'Customer'
)
  
```

To validate the table is working as expected by

- Using the code below into a **new cell (C)**, click the **run cell** button.

```

%%sql
SELECT * FROM CustomersOLTP
LIMIT 10
  
```

As you can see the result is the content of the Azure Cosmos DB transactional store.

The screenshot shows the Microsoft Azure Synapse Analytics workspace. At the top, there's a navigation bar with 'Microsoft Azure', 'Synapse Analytics', and a workspace name. Below it is a toolbar with icons for home, validate, publish, and preview features.

The main area displays two notebooks:

- Cosmos DB Notebook:** This notebook has a status of 'Ready'. It contains the following PySpark code:

```
1 %sql
2 CREATE TEMP VIEW customersOLTP USING cosmos.oltp OPTIONS (
3   spark.synapse.linkedService 'AdventureWorksSQL',
4   spark.cosmos.container 'Customer'
5 )
```

A red circle labeled 'B' is positioned over the code editor area.
- [141]:** This section shows the execution results of a SQL query:

```
1 %sql
2 SELECT * FROM customersOLTP
3 LIMIT 10
```

A red circle labeled 'C' is positioned over the results table.

At the bottom of the interface, there are tabs for 'View', 'Table' (which is selected), and 'Chart'. There are also links to 'View in monitoring' and 'Open Spark UI'.

Knowledge check

Question 1

When you want to switch to SparkSQL in a notebook, what is the first command to type?

- %%sparksql.
- %%spark.
- %%sql.

Question 2

What SparkSQL method reads data from the analytical store?

- cosmos.olap.
- cosmos.oltp.
- cosmos.db.

Summary

In this module, you have learned how to use the Synapse Spark to query the Azure Cosmos DB data made available by Azure Synapse Link.

Now you have completed this module, you'll be able to:

- Query the Azure Cosmos DB analytical store
- Perform simple aggregations with analytical store data
- Perform cross container queries in Azure Cosmos DB

- Perform complex queries with JSON data
- Work with the windowing function and other aggregations
- Write data back to the Azure Cosmos DB transactional store
- Read data from the transactional store

Query Azure Cosmos DB with SQL Serverless for Azure Synapse Analytics

Introduction

In this module, you will learn how to use Azure Synapse Analytics serverless SQL pools to query the Azure Cosmos DB data made available by Azure Synapse Link.

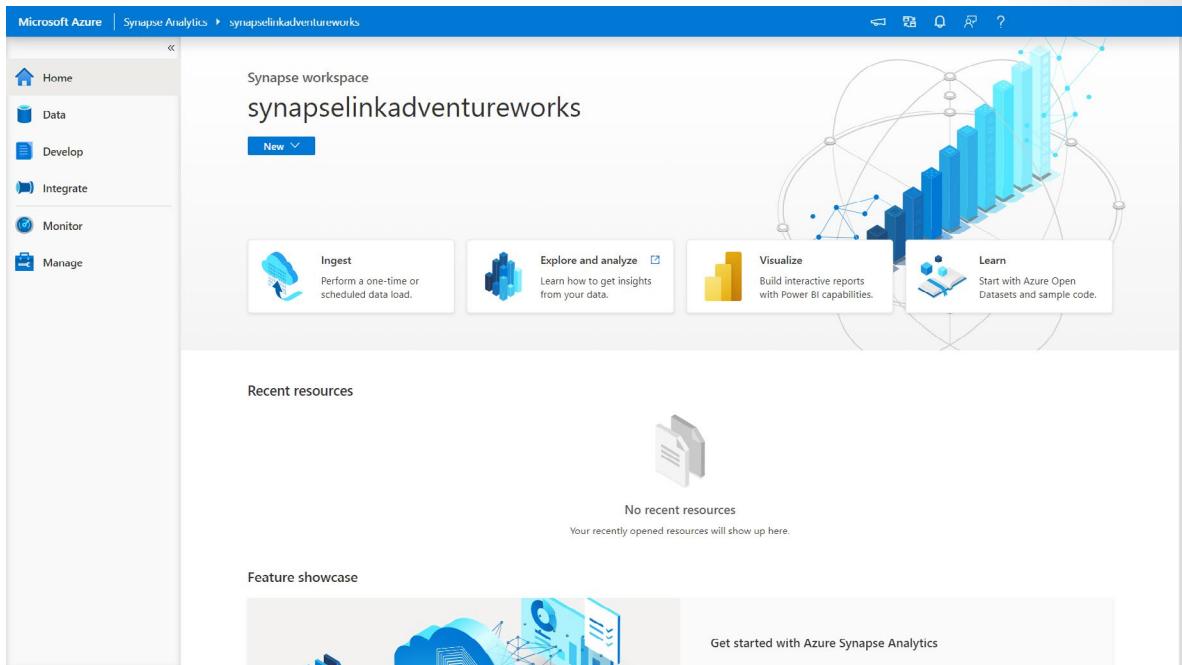
After completing this module, you'll be able to:

- Write basic queries against a single container
- Perform cross dataset queries in Cosmos DB
- Work with windowing function
- Perform complex queries with JSON data
- Visualize Azure Cosmos DB data in Power BI

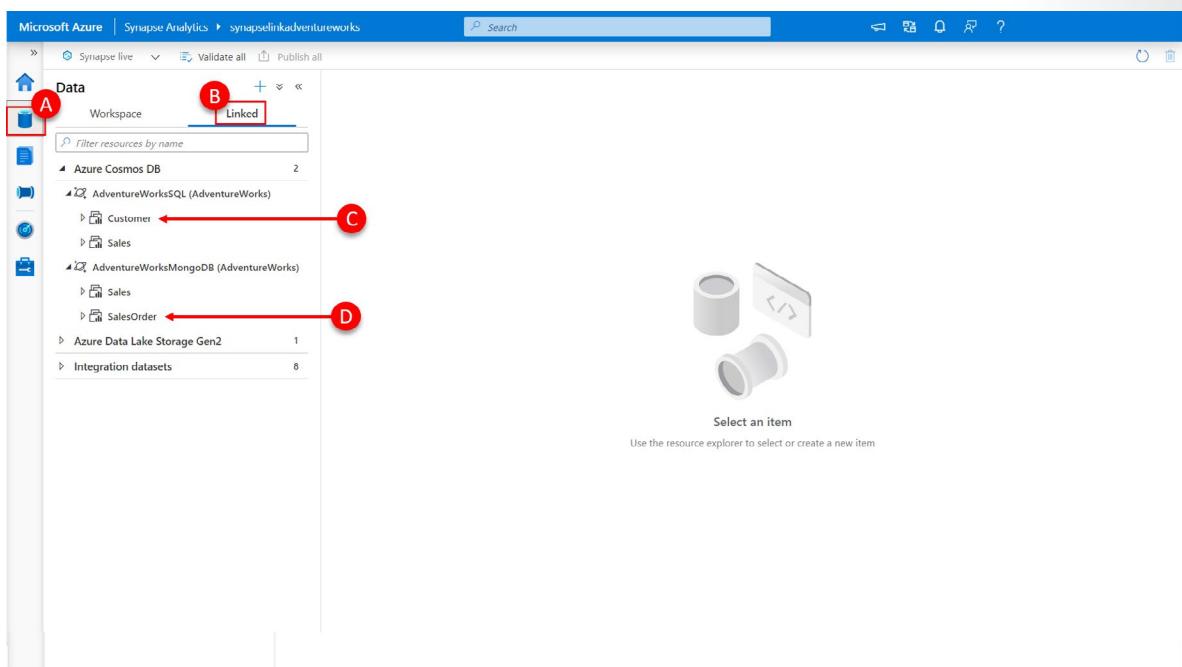
Write basic queries against a single container

We are going to be working the same two new containers (Customer and SalesOrder) we did in the previous module. The Customer and SalesOrder containers each contain example Adventure Works datasets of related customer profile records and sales order records respectively. These data sets reside in different Azure Cosmos DB accounts: the customer profile data resides in an Azure Cosmos DB Core (SQL) API account and the sales order data resides in Azure Cosmos DB API for MongoDB account, given that this data comes from distinct source systems. Adventure Works wants to use their available operational data to get insight into:

- What amount of revenue is coming from customers without completed profile data (no address details provided)
 - How sales order volume and revenue are distributed by city for those customers where they do have address details.
1. Connect to an Azure Synapse Workspace that has an Azure Synapse SQL Serverless instance, and an Azure Synapse Spark Pool.



2. In the left-hand menu, select **Data (A)**
3. Click on the **Linked tab** in the explorer view **(B)**
4. Expand the **AdventureWorksSQL** linked service to expose the **Customer** container
5. Expand the **AdventureWorksMongoDB** linked service to expose the **SalesOrder** container.



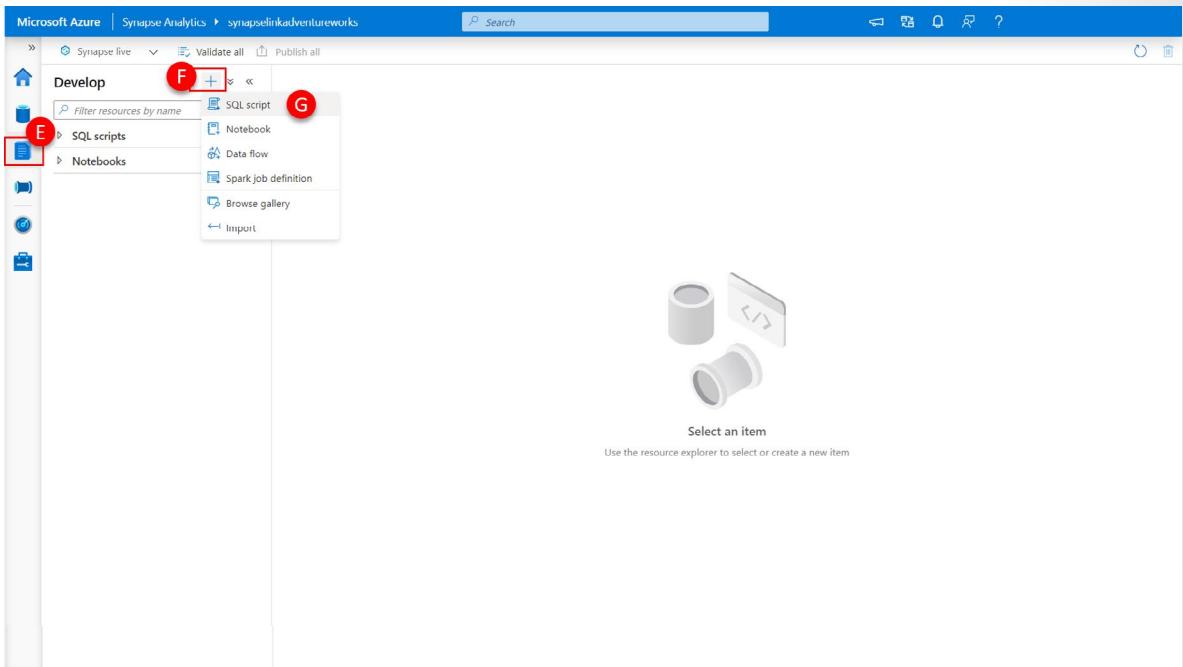
Here you can see additional two containers now visible in the data explorer view under the previously created linked service to our Azure Cosmos DB accounts (these containers already have Adventure Works data loaded into them) . The first, **Customer (C)**, has been created in the AdventureWorks database within the Azure Cosmos DB SQL API account and contains customer profile information.

The second, **SalesOrder (D)**, has been created the AdventureWorks database within the Azure Cosmos DB API for MongoDB account and contains sales order information.

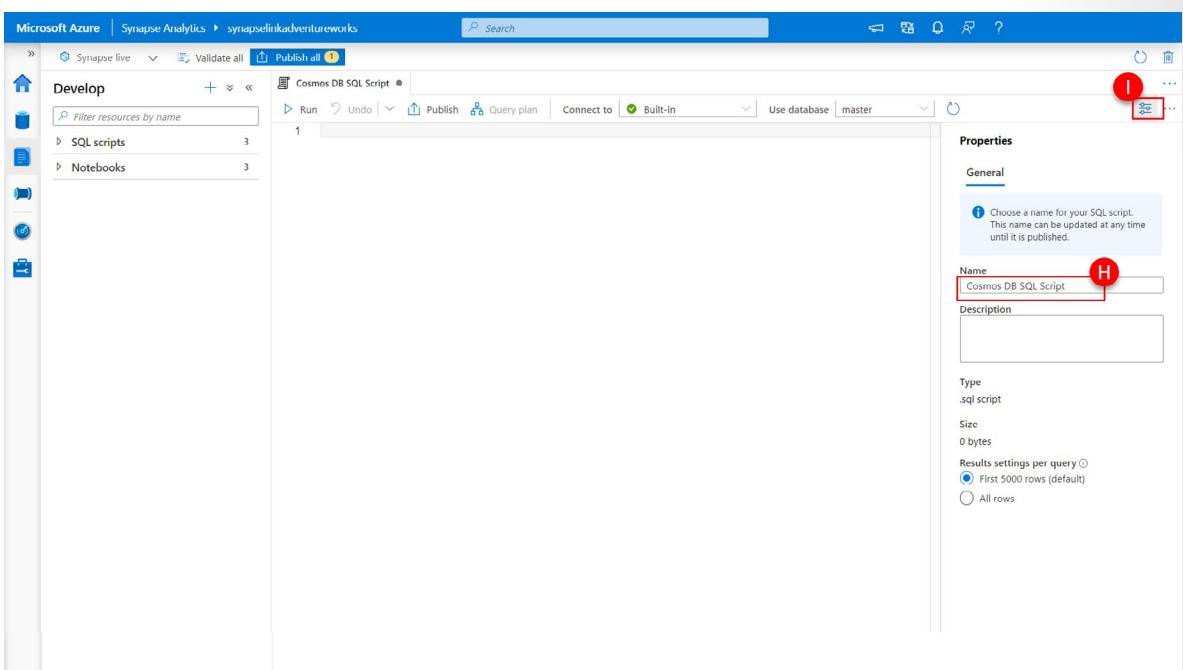
Lets run some queries to explore what is in the Customer container.

For reference here is a sample of a customer profile JSON document from the Customer container:

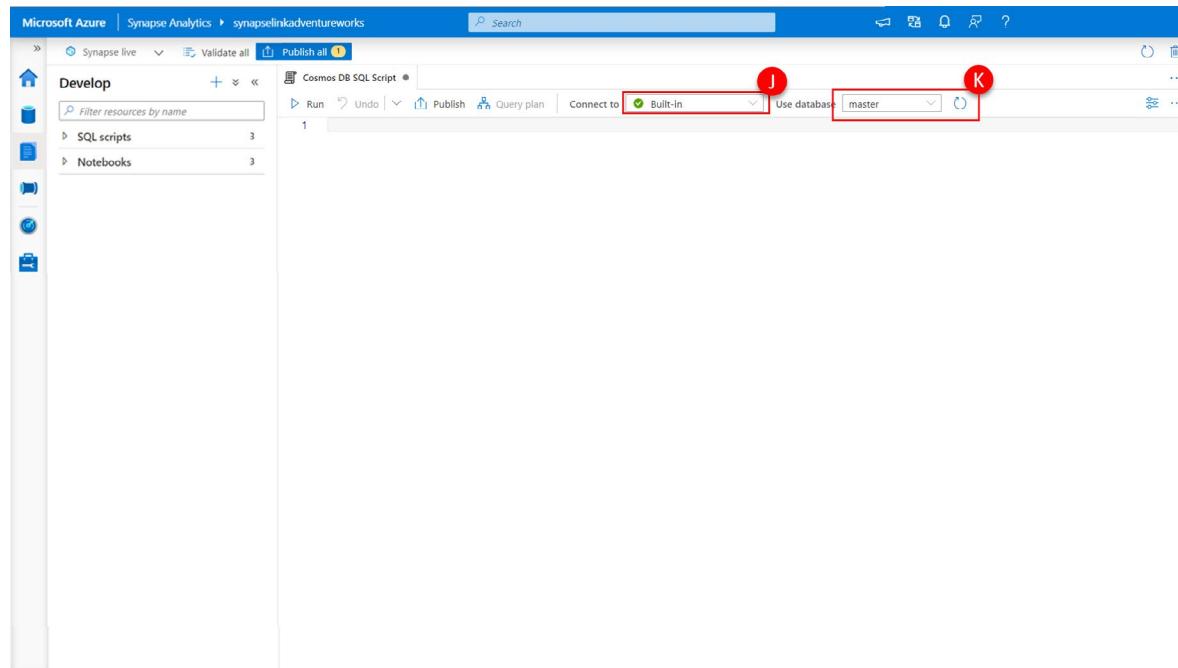
```
{  
  "id": "3E6A563E-9F01-46EA-BDE2-7C38FD02BE29",  
  "title": "",  
  "firstName": "Ruben",  
  "lastName": "Madan",  
  "emailAddress": "ruben8@adventure-works.com",  
  "phoneNumber": "1 (11) 500 555-0167",  
  "creationDate": "2013-12-20T00:00:00",  
  "address": {  
    "addressLine1": "59, boulevard Tremblay",  
    "addressLine2": "",  
    "city": "Paris",  
    "state": "75 ",  
    "country": "FR",  
    "zipCode": "75008"  
  },  
  "password": {  
    "hash": "L6rgOSf0NMOhruu/t/+tzWBzO0VNcyUL9qQf3GVW+9A=",  
    "salt": "76E5DBEB"  
  },  
  "_rid": "mP07AMBVkP0DAAAAAAA==",  
  "_self": "dbs/mP07AA==/colls/mP07AMBVkP0=/docs/mP07AMBVkP0DAAAAAAA==/",  
  "_etag": "\"c401ba71-0000-0800-0000-5fd2ff6e0000\"",  
  "_attachments": "attachments/",  
  "_ts": 1607663471  
}
```



6. In the left-hand menu, select **develop (E)**
7. Then click the “+” (F) to add a resource.
8. And then select **SQL Script (G)** to create a new SQL script. A new SQL Script will immediately be created within the Synapse Workspace.
9. Within the SQL script properties, provide an appropriate name **Cosmos DB SQL Script (H)**
10. And then click the **properties icon (I)** to close the properties blade.



Because we are going to utilize the built-in SQL Serverless pool, you can choose **built-in** from the **connect to:** drop down at the top of the query pane (J). You will see that the default database is now **master**.



Lets create a database in which we will store the objects we are going to query.

11. Paste the following Transact-SQL code into the **query pane (L)** and click the **run button at the top of the query pane (M)**.

```
```sql
CREATE DATABASE SynapseLinkDB
GO

USE SynapseLinkDB
GO
```
```


This will create a database named ****SynapseLinkDB**** and the select it for use within the subsequent session.

An alternative method of selecting this database for use is to click on the ****Use database (N)**** drop-down and selecting the database you wish to use, in our case ****SynapseLinkDB (O)****.

Let's also create a little more display real-estate by minimizing the

explorer blade by:

12. clicking << in the top right of the **blade (P)**.

We are now going to perform a SELECT operation against the Azure Cosmos DB analytical store using the OPENROWSET() function by:

13. Paste the following SQL into the query pane, making sure to **delete the previous SQL (Q)**.

```
SELECT TOP(10) *
FROM OPENROWSET('CosmosDB',
    'Account=adventureworksql;
Database=AdventureWorks;
Key=pRm30NZbE879vZa...Euw=='
    ,Customer) AS Customer
```

| Title | _rid | _ts | _etag | Id | LastName | FirstName | EmailAddress | PhoneNumber | CreationDate |
|-------|----------------|------------|-------------------|------------------|----------|-----------|-------------------|-------------------|------------------|
| JUL | mp07AMBvKp0... | 1607663471 | "c401d271-000... | 285FDBD7-54E... | Suarez | Julio | julio20@advent... | 1 (11) 500 555... | 2012-01-10T00... |
| JUL | mp07AMRVkP2... | 1607663471 | "c401f6072-000... | 936C50DR-7D6... | Harris | Randon | brandon14@ad... | 347-555-0181 | 2014-05-31T00... |
| JUL | mp07AMBvKp3... | 1607663471 | "c4017b72-000... | F4EAFF869-2D1... | Morris | Jada | jada10@advent... | 1 (11) 500 555... | 2014-03-30T00... |
| JUL | mp07AMBvKp0... | 1607663471 | "c401e772-000... | 5FB8B4AA-435... | Ross | Eric | eric12@advent... | 837-555-0153 | 2014-06-08T00... |
| THOM | mp07AMRVkP3... | 1607663471 | "c401a872-000... | 420092457-07C... | Thomas | Thomas | thomas70@adu... | 366-888-0161 | 2012-12-15T00... |

14. And then click **run**.

This will return the top (10) rows from the connected Azure Cosmos DB analytical store via Synapse Link

There are several parameters that need to be provided to the OPENROWSET function.

- The **provider name**: which in the case of the Cosmos DB analytical store is **CosmosDB**.
- The **provider string**: which in the case of the Cosmos DB analytical store is a connection string including the Azure Cosmos DB account name, the Database specifying the Azure Cosmos DB you wish to use and the Key for the account.
- The **table name**: which in the case of the Cosmos DB analytical store the container we wish to work with.

You will now be presented with a result set of the first 10 rows of a row-based representation of the documents contained within the Customer container's analytical store. This is held within an Azure

Cosmos Core (SQL) API account, so that data will be represented using the well-defined schema representation by default.

The top-level properties of the document are represented as columns with the associated property values as the value of the column. In the case that these values are primitive data types ("string", "integer", "float" etc.), the column will be apparently **typed (R)**, if these properties are embedded arrays or objects within the document, the column value will be a structure of these **embedded values (S)**.

In the case of our example the title, firstName, lastName, emailAddress, and phoneNumber properties are primitive strings and **assigned to their own columns (R)**, the address and password properties are both **embedded objects (S)**.

There is also the presence of **several Azure Cosmos DB system document properties (T)**. Azure Cosmos DB automatically has system properties such as _ts, _self, _attachments, _rid, and _etag associated with every document. These system document properties are seldom useful for analytical store query purposes, and will be ignored going forward.

Let us now create a view of our customers that we can use in our SQL queries in future.

15. Paste the following SQL into the query pane, making sure to delete the previous SQL statement.

```
CREATE VIEW Customers
AS
SELECT *
    FROM OPENROWSET('CosmosDB',
        'Account=adventureworks-sql;
Database=AdventureWorks;
Key=NZbE879vZa...Euw==';
Customer)
WITH
(
    CustomerId varchar(max) '$.id',
    Title varchar(max) '$.title',
    FirstName varchar(max) '$.firstName',
    LastName varchar(max) '$.lastName',
    EmailAddress varchar(max) '$.emailAddress',
    PhoneNumber varchar(max) '$.phoneNumber',
    AddressLine1 varchar(max) '$.address.addressLine1',
    AddressLine2 varchar(max) '$.address.addressLine2',
    City varchar(max) '$.address.city',
    Country varchar(max) '$.address.country',
    ZipCode varchar(max) '$.address.zipCode'
)
AS Customers
```

```

1 CREATE VIEW CustomerS
2 AS
3 SELECT *
4   FROM OPENROWSET('CosmosDB',
5     'Account=adventureworks-sql;Database=Adventureworks;Key=pRm30NZbE879vZaVS2JnTHhFkBPF',
6     'Customer')
7   (
8     CustomerId varchar(max) '$.id',
9     Title varchar(max) '$.title',
10    FirstName varchar(max) '$.firstName',
11    LastName varchar(max) '$.lastName',
12    EmailAddress varchar(max) '$.emailAddress',
13    PhoneNumber varchar(max) '$.phoneNumber',
14    AddressLine1 varchar(max) '$.address.addressLine1',
15    AddressLine2 varchar(max) '$.address.addressLine2',
16    City varchar(max) '$.address.city',
17    Country varchar(max) '$.address.country',
18    ZipCode varchar(max) '$.address.zipCode'
19  )
20  AS Customers
21
  
```

Results Messages

No results to show

Your query yielded no displayable results

00:00:03 Query executed successfully.

Here you will note that we have expanded the functionality of OPENROWSET function of the CREATE VIEW statement to include a **WITH clause (U)** that allows us to:

- Specify an alias for the column name (V), for example renaming the ID attribute to CustomerId in our example
- Specify the datatype of the underlying column store (W)
- Specify the JSON path of the property, the value of which to return in the specified column, this includes returning property values within embedded objects within the JSON. (X), for example "\$.address.zipCode" to access the zipCode property contained within the embedded address object.

16. Now query the view that you have just created

```
SELECT TOP(10) * FROM Customers
```

The screenshot shows the Microsoft Azure Synapse Analytics interface. A red circle labeled 'Z' highlights the 'Results' tab where the query results are displayed. The results are presented in a table format, showing 10 rows of customer data from the 'Customer' view. The columns include CustomerId, Title, FirstName, LastName, EmailAddress, PhoneNumber, AddressLine1, AddressLine2, City, Country, and ZipCode. The data is flattened from the original JSON documents.

| CustomerId | Title | FirstName | LastName | EmailAddress | PhoneNumber | AddressLine1 | AddressLine2 | City | Country | ZipCode |
|---------------------------------|-------|-----------|----------|-------------------|--------------------|---------------------|--------------|-------------|---------|---------|
| 285FDBD7-54EC-4F35-B436-4E19... | NULL | Julio | Suarez | julio20@advent... | 1 (11) 500 555-... | 2687 Apollo Way | NULL | Watford | GB | WA3 |
| 036C58DB-7D6F-42C1-8DBB-8E... | NULL | Brandon | Harris | brandon34@ad... | 347 555 0181 | 6369 Condor Pl... | NULL | Daly City | US | 94015 |
| F4EAF869-2D1E-4D70-B37D-FE8... | NULL | Jada | Morris | jada10@advent... | 1 (11) 500 555-... | Holzstr 1333 | NULL | Bottrop | DE | 46236 |
| 5FB848AA-A354-4D2E-A614-43... | NULL | Eric | Ross | eric12@advent... | 837-555-0153 | 1519 Birch Bark... | NULL | Chula Vista | US | 91910 |
| 420D3457-02C7-4609-9C25-7E78... | NULL | Thomas | Lewis | thomas79@adv... | 356-555-0161 | 1940 Nuty Drive | NULL | Lake Oswego | US | 97034 |
| 41294886-7F62-41F9-B2D3-5125... | NULL | Melody | Ramirez | melody17@adv... | 1 (11) 500 555-... | 2, rue de la C... | NULL | Morangis | FR | 91420 |
| 3FAAD072-EF57-41B2-B2C0-F2EA... | NULL | Sandra | Liu | sandra10@adv... | 1 (11) 500 555-... | 2, rue Pierre-De... | NULL | Morangis | FR | 91420 |
| 776BF142-3EBD-4D53-BD88-0FE5... | NULL | Ebony | Gill | ebony35@adv... | 327-555-0157 | 3235 Mi Casa Cu... | NULL | Birmingham | US | 35203 |
| 53302514 1883 4B7C 84FD C53F... | NULL | Gina | Torres | gina12@advnt... | 1 (11) 500 555 ... | 3841 Silver Osk... | NULL | Sunbury | AU | 3429 |
| 32AC5989-FA71-4FDC-A6E3-7F50... | NULL | Megan | Hall | megan26@adv... | 194-555-0177 | 6684 Galloway ... | NULL | El Cajon | US | 92020 |

00:00:14 Query executed successfully.

Here you can see the **results (Z)** from our Customers view, returning a flattened columnar view of the JSON documents that reside in the customer container within the Azure Cosmos DB SQL API account.

Perform cross container queries

Let's explore the second of our two containers; the SalesOrder container. This container hosts the AdventureWorks database within the Azure Cosmos DB API for MongoDB and contains sales order information.

For reference here is a sample of a sales order JSON document from the SalesOrder container:

```
{  
    "_id" : "F0FF0BA8-CEF4-454C-9173-FA7C546F7179",  
    "customerId" : "A852CB99-DAA1-4348-A86F-622D4478A7D0",  
    "orderDate" : "2013-07-11T00:00:00",  
    "shipDate" : "2013-07-18T00:00:00",  
    "details" : [  
        {  
            "sku" : "LJ-0192-M",  
            "name" : "Long-Sleeve Logo Jersey, M",  
            "price" : 49.99,  
            "quantity" : 1  
        },  
        {  
            "sku" : "SH-W890-M",  
            "name" : "Women's Mountain Shorts, M",  
            "price" : 69.99,  
            "quantity" : 1  
        }  
    ]}
```

}

Paste the following SQL into the query pane.

```
SELECT top 10 *
  FROM OPENROWSET('CosmosDB',
                   'Account=adventureworks-mongodb;Database=Adventure-
Works;Key=v2mtZ85W0AMCv1ZrY7j...4g==',
                   SalesOrder) As SalesOrders
```

The screenshot shows the Microsoft Azure Synapse Analytics workspace interface. The top navigation bar includes 'Microsoft Azure' and 'Synapse Analytics > synapsealinkadventureworks'. A search bar is at the top right. Below the navigation is a toolbar with icons for 'Synapse live', 'Validate all', 'Publish all', and a 'Script' tab selected. The main area contains a 'Cosmos DB SQL Script' editor with the following code:

```
1 SELECT top 10 *
2   FROM OPENROWSET('CosmosDB',
3     'Account=adventureworks-mongodb;Database=Adventureworks;Key=v2mtZB5W@AHcv1ZrY7jMU0WpfBT',
4     SalesOrder) As SalesOrders
5
```

Below the script are tabs for 'Results' and 'Messages'. The 'Results' tab is active, showing a table view with the following data:

| ShipDate | _id | OrderDate | CustomerId | Details |
|------------|-----------------------------------|-------------------------------------|-----------------------------------|--|
| 5MEUH... | {"string": "2012-12-21T00:00:00"} | {"string": "F02FB90E-8987-4F44-A... | {"string": "2012-12-14T00:00:00"} | {"array": [{"object": {"sku": {"string": "BK-M68B-38"}, "name": {"string": "Mountain-200 Black, 38"}, "price": 100}, {"object": {"sku": {"string": "FR-R38B-52"}, "name": {"string": "LL Road Frame - Black, 52"}, "price": 150}, {"object": {"sku": {"string": "CA-1098"}, "name": {"string": "AWC Logo Cap"}, "price": 5}, {"object": {"sku": {"string": "LI-0192-M"}, "name": {"string": "Long-Sleeve Logo Jersey, M"}, "price": 50}, {"object": {"sku": {"string": "BK-R03R-44"}, "name": {"string": "Road 150 Red, 44"}, "price": 120}, {"object": {"sku": {"string": "PK-7098"}, "name": {"string": "Patch Kit 8 Patches"}, "price": 10}, {"object": {"sku": {"string": "FE-6654"}, "name": {"string": "Fender Set - Mountain"}, "price": 80}, {"object": {"sku": {"string": "SH-W090-L"}, "name": {"string": "Women's Mountain Shorts, L"}, "price": 60}, {"object": {"sku": {"string": "TI-T723"}, "name": {"string": "Touring Tire"}, "price": 150}, {"object": {"sku": {"string": "FR-M215-40"}, "name": {"string": "LL Mountain Frame - Silver, 40"}, "price": 400}], "total": 1000}} |
| U2REMT... | {"string": "2012-06-06T00:00:00"} | {"string": "F09A56DC-617D-4232... | {"string": "2012-05-30T00:00:00"} | {"array": [{"object": {"sku": {"string": "BK-M68B-38"}, "name": {"string": "Mountain-200 Black, 38"}, "price": 100}, {"object": {"sku": {"string": "FR-R38B-52"}, "name": {"string": "LL Road Frame - Black, 52"}, "price": 150}, {"object": {"sku": {"string": "CA-1098"}, "name": {"string": "AWC Logo Cap"}, "price": 5}, {"object": {"sku": {"string": "LI-0192-M"}, "name": {"string": "Long-Sleeve Logo Jersey, M"}, "price": 50}, {"object": {"sku": {"string": "BK-R03R-44"}, "name": {"string": "Road 150 Red, 44"}, "price": 120}, {"object": {"sku": {"string": "PK-7098"}, "name": {"string": "Patch Kit 8 Patches"}, "price": 10}, {"object": {"sku": {"string": "FE-6654"}, "name": {"string": "Fender Set - Mountain"}, "price": 80}, {"object": {"sku": {"string": "SH-W090-L"}, "name": {"string": "Women's Mountain Shorts, L"}, "price": 60}, {"object": {"sku": {"string": "TI-T723"}, "name": {"string": "Touring Tire"}, "price": 150}, {"object": {"sku": {"string": "FR-M215-40"}, "name": {"string": "LL Mountain Frame - Silver, 40"}, "price": 400}], "total": 1000}} |
| g2NkIt... | {"string": "2013-10-30T00:00:00"} | {"string": "FOCC8368-666C-4A24-... | {"string": "2013-10-23T00:00:00"} | {"array": [{"object": {"sku": {"string": "BK-M68B-38"}, "name": {"string": "Mountain-200 Black, 38"}, "price": 100}, {"object": {"sku": {"string": "FR-R38B-52"}, "name": {"string": "LL Road Frame - Black, 52"}, "price": 150}, {"object": {"sku": {"string": "CA-1098"}, "name": {"string": "AWC Logo Cap"}, "price": 5}, {"object": {"sku": {"string": "LI-0192-M"}, "name": {"string": "Long-Sleeve Logo Jersey, M"}, "price": 50}, {"object": {"sku": {"string": "BK-R03R-44"}, "name": {"string": "Road 150 Red, 44"}, "price": 120}, {"object": {"sku": {"string": "PK-7098"}, "name": {"string": "Patch Kit 8 Patches"}, "price": 10}, {"object": {"sku": {"string": "FE-6654"}, "name": {"string": "Fender Set - Mountain"}, "price": 80}, {"object": {"sku": {"string": "SH-W090-L"}, "name": {"string": "Women's Mountain Shorts, L"}, "price": 60}, {"object": {"sku": {"string": "TI-T723"}, "name": {"string": "Touring Tire"}, "price": 150}, {"object": {"sku": {"string": "FR-M215-40"}, "name": {"string": "LL Mountain Frame - Silver, 40"}, "price": 400}], "total": 1000}} |
| ICQIgtQ... | {"string": "2013-07-18T00:00:00"} | {"string": "F0FF08AB-CEF4-454C-9... | {"string": "2013-07-11T00:00:00"} | {"array": [{"object": {"sku": {"string": "BK-M68B-38"}, "name": {"string": "Mountain-200 Black, 38"}, "price": 100}, {"object": {"sku": {"string": "FR-R38B-52"}, "name": {"string": "LL Road Frame - Black, 52"}, "price": 150}, {"object": {"sku": {"string": "CA-1098"}, "name": {"string": "AWC Logo Cap"}, "price": 5}, {"object": {"sku": {"string": "LI-0192-M"}, "name": {"string": "Long-Sleeve Logo Jersey, M"}, "price": 50}, {"object": {"sku": {"string": "BK-R03R-44"}, "name": {"string": "Road 150 Red, 44"}, "price": 120}, {"object": {"sku": {"string": "PK-7098"}, "name": {"string": "Patch Kit 8 Patches"}, "price": 10}, {"object": {"sku": {"string": "FE-6654"}, "name": {"string": "Fender Set - Mountain"}, "price": 80}, {"object": {"sku": {"string": "SH-W090-L"}, "name": {"string": "Women's Mountain Shorts, L"}, "price": 60}, {"object": {"sku": {"string": "TI-T723"}, "name": {"string": "Touring Tire"}, "price": 150}, {"object": {"sku": {"string": "FR-M215-40"}, "name": {"string": "LL Mountain Frame - Silver, 40"}, "price": 400}], "total": 1000}} |
| ISMERIN... | {"string": "2011-06-13T00:00:00"} | {"string": "F01D190B-5A6C-42A5 ... | {"string": "2011-06-06T00:00:00"} | {"array": [{"object": {"sku": {"string": "BK-M68B-38"}, "name": {"string": "Mountain-200 Black, 38"}, "price": 100}, {"object": {"sku": {"string": "FR-R38B-52"}, "name": {"string": "LL Road Frame - Black, 52"}, "price": 150}, {"object": {"sku": {"string": "CA-1098"}, "name": {"string": "AWC Logo Cap"}, "price": 5}, {"object": {"sku": {"string": "LI-0192-M"}, "name": {"string": "Long-Sleeve Logo Jersey, M"}, "price": 50}, {"object": {"sku": {"string": "BK-R03R-44"}, "name": {"string": "Road 150 Red, 44"}, "price": 120}, {"object": {"sku": {"string": "PK-7098"}, "name": {"string": "Patch Kit 8 Patches"}, "price": 10}, {"object": {"sku": {"string": "FE-6654"}, "name": {"string": "Fender Set - Mountain"}, "price": 80}, {"object": {"sku": {"string": "SH-W090-L"}, "name": {"string": "Women's Mountain Shorts, L"}, "price": 60}, {"object": {"sku": {"string": "TI-T723"}, "name": {"string": "Touring Tire"}, "price": 150}, {"object": {"sku": {"string": "FR-M215-40"}, "name": {"string": "LL Mountain Frame - Silver, 40"}, "price": 400}], "total": 1000}} |
| 4M7rgt... | {"string": "2014-02-23T00:00:00"} | {"string": "F15A9838-4105-4BF6-B... | {"string": "2014-02-16T00:00:00"} | {"array": [{"object": {"sku": {"string": "BK-M68B-38"}, "name": {"string": "Mountain-200 Black, 38"}, "price": 100}, {"object": {"sku": {"string": "FR-R38B-52"}, "name": {"string": "LL Road Frame - Black, 52"}, "price": 150}, {"object": {"sku": {"string": "CA-1098"}, "name": {"string": "AWC Logo Cap"}, "price": 5}, {"object": {"sku": {"string": "LI-0192-M"}, "name": {"string": "Long-Sleeve Logo Jersey, M"}, "price": 50}, {"object": {"sku": {"string": "BK-R03R-44"}, "name": {"string": "Road 150 Red, 44"}, "price": 120}, {"object": {"sku": {"string": "PK-7098"}, "name": {"string": "Patch Kit 8 Patches"}, "price": 10}, {"object": {"sku": {"string": "FE-6654"}, "name": {"string": "Fender Set - Mountain"}, "price": 80}, {"object": {"sku": {"string": "SH-W090-L"}, "name": {"string": "Women's Mountain Shorts, L"}, "price": 60}, {"object": {"sku": {"string": "TI-T723"}, "name": {"string": "Touring Tire"}, "price": 150}, {"object": {"sku": {"string": "FR-M215-40"}, "name": {"string": "LL Mountain Frame - Silver, 40"}, "price": 400}], "total": 1000}} |
| 2yN0U1... | {"string": "2013-11-25T00:00:00"} | {"string": "F186427E-8374-431A-... | {"string": "2013-11-18T00:00:00"} | {"array": [{"object": {"sku": {"string": "BK-M68B-38"}, "name": {"string": "Mountain-200 Black, 38"}, "price": 100}, {"object": {"sku": {"string": "FR-R38B-52"}, "name": {"string": "LL Road Frame - Black, 52"}, "price": 150}, {"object": {"sku": {"string": "CA-1098"}, "name": {"string": "AWC Logo Cap"}, "price": 5}, {"object": {"sku": {"string": "LI-0192-M"}, "name": {"string": "Long-Sleeve Logo Jersey, M"}, "price": 50}, {"object": {"sku": {"string": "BK-R03R-44"}, "name": {"string": "Road 150 Red, 44"}, "price": 120}, {"object": {"sku": {"string": "PK-7098"}, "name": {"string": "Patch Kit 8 Patches"}, "price": 10}, {"object": {"sku": {"string": "FE-6654"}, "name": {"string": "Fender Set - Mountain"}, "price": 80}, {"object": {"sku": {"string": "SH-W090-L"}, "name": {"string": "Women's Mountain Shorts, L"}, "price": 60}, {"object": {"sku": {"string": "TI-T723"}, "name": {"string": "Touring Tire"}, "price": 150}, {"object": {"sku": {"string": "FR-M215-40"}, "name": {"string": "LL Mountain Frame - Silver, 40"}, "price": 400}], "total": 1000}} |
| i5Nzk... | {"string": "2013-10-16T00:00:00"} | {"string": "F1C32979-C664-40F1-9... | {"string": "2013-10-11T00:00:00"} | {"array": [{"object": {"sku": {"string": "BK-M68B-38"}, "name": {"string": "Mountain-200 Black, 38"}, "price": 100}, {"object": {"sku": {"string": "FR-R38B-52"}, "name": {"string": "LL Road Frame - Black, 52"}, "price": 150}, {"object": {"sku": {"string": "CA-1098"}, "name": {"string": "AWC Logo Cap"}, "price": 5}, {"object": {"sku": {"string": "LI-0192-M"}, "name": {"string": "Long-Sleeve Logo Jersey, M"}, "price": 50}, {"object": {"sku": {"string": "BK-R03R-44"}, "name": {"string": "Road 150 Red, 44"}, "price": 120}, {"object": {"sku": {"string": "PK-7098"}, "name": {"string": "Patch Kit 8 Patches"}, "price": 10}, {"object": {"sku": {"string": "FE-6654"}, "name": {"string": "Fender Set - Mountain"}, "price": 80}, {"object": {"sku": {"string": "SH-W090-L"}, "name": {"string": "Women's Mountain Shorts, L"}, "price": 60}, {"object": {"sku": {"string": "TI-T723"}, "name": {"string": "Touring Tire"}, "price": 150}, {"object": {"sku": {"string": "FR-M215-40"}, "name": {"string": "LL Mountain Frame - Silver, 40"}, "price": 400}], "total": 1000}} |
| 0MjEq... | {"string": "2013-10-12T00:00:00"} | {"string": "F1F66421-B091-4504-B... | {"string": "2013-10-05T00:00:00"} | {"array": [{"object": {"sku": {"string": "BK-M68B-38"}, "name": {"string": "Mountain-200 Black, 38"}, "price": 100}, {"object": {"sku": {"string": "FR-R38B-52"}, "name": {"string": "LL Road Frame - Black, 52"}, "price": 150}, {"object": {"sku": {"string": "CA-1098"}, "name": {"string": "AWC Logo Cap"}, "price": 5}, {"object": {"sku": {"string": "LI-0192-M"}, "name": {"string": "Long-Sleeve Logo Jersey, M"}, "price": 50}, {"object": {"sku": {"string": "BK-R03R-44"}, "name": {"string": "Road 150 Red, 44"}, "price": 120}, {"object": {"sku": {"string": "PK-7098"}, "name": {"string": "Patch Kit 8 Patches"}, "price": 10}, {"object": {"sku": {"string": "FE-6654"}, "name": {"string": "Fender Set - Mountain"}, "price": 80}, {"object": {"sku": {"string": "SH-W090-L"}, "name": {"string": "Women's Mountain Shorts, L"}, "price": 60}, {"object": {"sku": {"string": "TI-T723"}, "name": {"string": "Touring Tire"}, "price": 150}, {"object": {"sku": {"string": "FR-M215-40"}, "name": {"string": "LL Mountain Frame - Silver, 40"}, "price": 400}], "total": 1000}} |
| ZBNjY... | {"string": "2013-08-07T00:00:00"} | {"string": "F47F8A66-4668-46E7-B... | {"string": "2013-07-31T00:00:00"} | {"array": [{"object": {"sku": {"string": "BK-M68B-38"}, "name": {"string": "Mountain-200 Black, 38"}, "price": 100}, {"object": {"sku": {"string": "FR-R38B-52"}, "name": {"string": "LL Road Frame - Black, 52"}, "price": 150}, {"object": {"sku": {"string": "CA-1098"}, "name": {"string": "AWC Logo Cap"}, "price": 5}, {"object": {"sku": {"string": "LI-0192-M"}, "name": {"string": "Long-Sleeve Logo Jersey, M"}, "price": 50}, {"object": {"sku": {"string": "BK-R03R-44"}, "name": {"string": "Road 150 Red, 44"}, "price": 120}, {"object": {"sku": {"string": "PK-7098"}, "name": {"string": "Patch Kit 8 Patches"}, "price": 10}, {"object": {"sku": {"string": "FE-6654"}, "name": {"string": "Fender Set - Mountain"}, "price": 80}, {"object": {"sku": {"string": "SH-W090-L"}, "name": {"string": "Women's Mountain Shorts, L"}, "price": 60}, {"object": {"sku": {"string": "TI-T723"}, "name": {"string": "Touring Tire"}, "price": 150}, {"object": {"sku": {"string": "FR-M215-40"}, "name": {"string": "LL Mountain Frame - Silver, 40"}, "price": 400}], "total": 1000}} |

A red circle with the letter 'A' is positioned over the first row of the table. A red circle with the letter 'B' is positioned over the second row of the table.

Click run.

You are presented with a result set of the first 10 rows. It is a row-based representation of the documents contained within the SalesOrder container's analytical store using Azure Cosmos Core API account for MongoDB. As a result, the data is represented using the full fidelity schema representation by default.

All top-level properties of the document are represented as columns with the associated property values. All properties are **represented as a structure of the type of values assigned to the properties; and the values themselves, (A) and (B)**. For complex types such as objects and arrays, they will remain embedded within the structure, and expanded to include type encapsulation of each of their property values.

In this example, ignoring the system document properties for now, the `_id`, `customerId`, `orderDate`, and `shipDate` are all strings and have a **type encapsulation of string (A)**. The **details property is an embedded array (B)**, in turn with embedded properties `sku`, `name`, `price`, and `quantity` of each array element object.

Let's use what we observed here, and what we learned earlier about using OPENROWSET to combine this data with our Customers view we created previously to create a new SalesOrders view by creating the following script

```
CREATE VIEW SalesOrders
```

```

SELECT SalesOrderId, SalesOrders.customerId CustomerId,
       CONVERT(date,orderDate) as OrderDate, CONVERT(date,shipdate) AS ShipDate,
       Customers.Country, Customers.City
  FROM OPENROWSET('CosmosDB',
                   'Account=adventureworks-mongodb;Database=Adventure-
Works;Key=v2mtZ85W0AMCv1Zr...D3v4g==',
                   SalesOrder)
  WITH
  (
      SalesOrderId varchar(max) '$._id.string',
      customerId varchar(max) '$.customerId.string',
      orderDate varchar(max) '$.orderDate.string',
      shipDate varchar(max) '$.shipDate.string'
  )
  As SalesOrders
 INNER JOIN Customers
   On SalesOrders.customerId = Customers.CustomerId

```

The screenshot shows the Microsoft Azure Synapse Analytics interface. A query is being written in the 'Cosmos DB SQL Script' view. The code is annotated with red circles:

- A:** Points to the 'AS' keyword.
- B:** Points to the 'CustomerID' column in the 'SELECT' clause.
- C:** Points to the 'WITH' clause.
- D:** Points to the '\$.shipDate.string' suffix, indicating the full fidelity schema path.
- E:** Points to the 'CustomerId' key in the 'ON' clause of the 'INNER JOIN'.
- F:** Points to the 'OrderDate' column in the 'SELECT' clause.

The query itself is:

```

CREATE VIEW SalesOrders
AS
SELECT SalesOrderId, SalesOrders.customerId CustomerId,
       CONVERT(date,orderDate) as OrderDate, CONVERT(date,shipdate) AS ShipDate,
       Customers.Country, Customers.City
  FROM OPENROWSET('CosmosDB',
                   'Account=adventureworks-mongodb;Database=AdventureWorks;Key=v2mtZ85W0AMCv1Zr...D3v4g==',
                   SalesOrder)
  WITH
  (
      SalesOrderId varchar(max) '$._id.string',
      customerId varchar(max) '$.customerId.string',
      orderDate varchar(max) '$.orderDate.string',
      shipDate varchar(max) '$.shipDate.string'
  )
  As SalesOrders
 INNER JOIN Customers
   On SalesOrders.customerId = Customers.CustomerId

```

Here you can see we are using the **WITH clause (C)** again to specify the path to our property values, notably **including the data type suffix (D)** of the property as part of the path to the property in order to access the values when using the full fidelity schema. In this example, we need to specify "\$.shipDate.string" rather than just "\$.shipDate".

We are joining it back to our Customers view using the **CustomerId key present in both (E)**, and doing some explicit type conversion for our date values using the CONVERT function to project them as SQL date data types.

Let see what this view returns by:

```
SELECT top 10 * FROM SalesOrders
```

The screenshot shows the Microsoft Azure Synapse Analytics interface. In the top navigation bar, 'Synapse live' is selected. Below it, a 'Cosmos DB SQL Script' tab is active. The main area contains a query pane with the following SQL code:

```

1
2 SELECT Top 10 * FROM SalesOrders

```

Below the query pane, there are tabs for 'Results' and 'Messages'. The 'Results' tab is selected, showing a table view of the query results. The table has columns: SalesOrderID, CustomerID, OrderDate, ShipDate, Country, and City. The results show 10 rows of data from the SalesOrders view, flattened into a columnar format. A red box highlights this table output.

| SalesOrderID | CustomerID | OrderDate | ShipDate | Country | City |
|---------------------------------|---------------------------------|-----------------------------|-----------------------------|---------|----------------|
| 967E2756-5A9C-4F7C-A2D9-22C... | 00015962-05D8-46E5-A2D9-AE6... | 2013-08-22T00:00:00.0000000 | 2013-08-29T00:00:00.0000000 | AU | East Brisbane |
| A1C87FDD-1823-463A-9787-E46... | 000C5E5B-E5B6-4958-B3C0-BB2... | 2011-10-13T00:00:00.0000000 | 2011-10-20T00:00:00.0000000 | US | Burbank |
| 33D323A1-1523-449D-86D9-855... | 00182FAD-5C64-4A77-81C8-DEF... | 2014-06-30T00:00:00.0000000 | 2014-07-07T00:00:00.0000000 | US | Redwood City |
| 7FB380E2-A2E5-46D0-8EF3-C6EF... | 001C8C0B-9891-47A5-A198-877... | 2014-03-17T00:00:00.0000000 | 2014-03-24T00:00:00.0000000 | DE | Hamburg |
| E70DAGA1-AF1B-4B71-B7B6-816... | 00307EA7-G25F-4C5D-A99F-32B... | 2013-11-19T00:00:00.0000000 | 2013-11-26T00:00:00.0000000 | AU | Melbourne |
| 6A7C863B-3F98-426D-9F33-719... | 0031725D-6D6C-40C8-943F-4C2... | 2013-11-13T00:00:00.0000000 | 2013-11-20T00:00:00.0000000 | DE | Paderborn |
| F01C55BD-29F3-4D56-8A7-AB... | 0052B816-95B9-45DE-964A-C9B... | 2014-05-20T00:00:00.0000000 | 2014-05-27T00:00:00.0000000 | US | Seattle |
| 6308CA3F-FFA0-4E58-8B78-426... | 0092801F-9D23-4818-A8AD-3F6... | 2014-06-19T00:00:00.0000000 | 2014-06-26T00:00:00.0000000 | US | Woodland Hills |
| 9E46328D-83C9-4428-8561-FF71... | 00A61365-0110-4469-81C9-3F0E... | 2013-07-08T00:00:00.0000000 | 2013-07-15T00:00:00.0000000 | DE | Berlin |
| 5528F49A-D031-4C64-8FF1-E04... | 00D81835-F5F7-4970-B650-03C... | 2013-11-21T00:00:00.0000000 | 2013-11-28T00:00:00.0000000 | GB | Billericay |

At the bottom of the results pane, a message indicates: '00:00:18 Query executed successfully.'

Here you can see the results from our SalesOrders view, returning a flattened columnar view of the JSON documents that reside in the SalesOrder container within the Azure Cosmos DB API for MongoDB account.

Perform complex queries with JSON data

Let us focus on extracting the data from sales order details for now. To do that we are going to want to look at the SalesOrderId (contained within the `_id` property of the document) and the details property that contains the array of sales order details.

Paste the following SQL into the query pane.

```

SELECT TOP (10) SalesOrderId, details
FROM OPENROWSET('CosmosDB',
    'Account=adventureworks-mongodb;Database=Adventure-
    Works;Key=v2mtZ85W0AMCv1ZrY7jMUOWpfBTi1BrUz0Y3Rwmvj9SXSSIKDU7EQVu5kdEM-
    cwAQfvJBnmHSMxy50c3gD3v4g==',
    SalesOrder)
WITH
(
    SalesOrderId varchar(max) '$._id',
    details varchar(max) '$.details'
) As SalesOrders

```

The screenshot shows the Microsoft Azure Synapse Analytics interface. In the top navigation bar, 'Synapse live' is selected. Below it, the query editor has tabs for 'Run', 'Undo', 'Publish', 'Query plan', 'Connect to', 'Built-in', 'Use database', and 'SynapseLinkDB'. The main area contains a SQL script:

```

1 SELECT TOP(10) SalesOrderId, details
2   FROM OPENROWSET('CosmosDB',
3     'Account=adventureworks-mongodb;Database=Adventureworks;Key=v2mtZ85W0AMCv1ZrY7jMUOWpfBTi1BrUz0Y3Rwmvj9SXSSIKDU7EQVu5kdEM-
4     cWAQfvJBnmHSMyxy50c3gD3v4g==',
5     SalesOrder)
6   WITH
7     ( SalesOrderId varchar(max) '$._id.string',
8       details varchar(max) '$.details.array'
9     ) As SalesOrders

```

The results pane shows the output of the query. Column A points to the 'SalesOrderId' column, which contains JSON documents. Column B points to the 'details' column, which contains arrays of JSON objects. Column C points to one element within an array. Column D points to another element within an array. Column E points to the array itself.

Annotations:

- A**: Points to the 'SalesOrderId' column header.
- B**: Points to the 'details' column header.
- C**: Points to a single element within an array.
- D**: Points to another element within the same array.
- E**: Points to the array itself.

At the bottom of the results pane, a message says: "000007 Query executed successfully."

Click **run**.

Here we can see the **SalesOrderId column (A)** is returning a JSON fragment including the data type of the document _id property, in this case string along with the property values. The **details column (B)** is also returning a JSON fragment in this case indicating that the data type is an **array (C)** and that it contains **multiple array elements (D)**.

Let us now access these properties by correctly specifying the path including the type suffix for each property (in this case "string" for _id and "array" for the details attributes) by:

Paste the following SQL into the query pane.

```

SELECT TOP(10) SalesOrderId, details
  FROM OPENROWSET('CosmosDB',
    'Account=adventureworks-mongodb;Database=Adventure-
Works;Key=v2mtZ85W0AMCv1ZrY7jMUOWpfBTi1BrUz0Y3Rwmvj9SXSSIKDU7EQVu5kdEM-
    cWAQfvJBnmHSMyxy50c3gD3v4g==',
    SalesOrder)
   WITH
      ( SalesOrderId varchar(max) '$._id.string',
        details varchar(max) '$.details.array'
      ) As SalesOrderDetails

```

The screenshot shows the Microsoft Azure Synapse Analytics query editor. A query is being run against a database named 'SynapseLinkAdventureworks'. The query uses OPENROWSET to connect to a CosmosDB table named 'SalesOrder' and then applies an OPENJSON function to extract data into a temporary table 'SalesOrderDetails'. The results pane displays the extracted data, showing the SalesOrderId and the corresponding JSON array in the Details column.

Annotations in the screenshot:

- F**: Points to the 'Run' button at the top left.
- G**: Points to the 'Results' tab at the bottom left.
- E**: Points to the array designator '[]' in the query code.
- H**: Points to the first element of a JSON array in the 'Details' column.
- I**: Points to the suffix 'object type suffix' in the query code.

Click **run**.

Now we can see that the SalesOrderId column contains just the appropriate value and the details column now just contains the JSON array itself starting with the array designator "[" and ending with "]" and can contain **multiple array elements (H)** of object type as indicated in the JSON with **object type suffix (I)**.

Let now create a SalesOrderDetails view by:

Paste the following SQL into the query pane.

```

CREATE VIEW SalesOrderDetails
AS
SELECT SalesOrderId, SalesOrderArray.[key]+1 as SalesOrderLine, SKUCode,
SKUName, Price, Quantity
FROM OPENROWSET('CosmosDB',
    'Account=adventureworks-mongodb;Database=Adventure-
    Works;Key=v2mtZ85W0AMCv1ZrY7jMUOWpfBTi1BrUz0Y3Rwmvj9SXSSIKDU7EQVu5kdEM-
    cwAQfvJBnmHSMxy50c3gD3v4g==',
    SalesOrder)
    WITH
        ( SalesOrderId varchar(max) '$._id.string',
            details varchar(max) '$.details.array'
        ) As SalesOrders
CROSS APPLY OPENJSON(SalesOrders.details) AS SalesOrderArray
CROSS APPLY OPENJSON(SalesOrderArray.[value])
    WITH
        (SKUCode varchar(max) '$.object.sku.string',
        SKUName varchar(max) '$.object.name.string',
        Price decimal(10,4) '$.object.price.float64',
        Quantity int '$.object.quantity.int32'
    ) As SalesOrderDetails

```

The first OPENJSON clause provides the SalesOrders.details value we extracted in using the WITH clause of the OPENROWSET. When you call OPENJSON without a WITH clause and provide it with a JSON

fragment that represents an array (as is the case in our example) the function returns a table with the following columns:

- Key - A value that contains the zero-based index of the element in the specified array.
- Value - An nvarchar(max) value that contains the value of the property itself. This will be the object value for each of the array elements in our example.
- Type - An int value that contains the type of the value, which we don't use in our example.

The second OPENJSON clause is provided with an input value of the value returned by the first OPENJSON clause, in our example the JSON fragment that represents the object of each element within the array. In this case, we use the WITH clause to further specify the column alias, and the column data type and the element path we want to access (remembering to include the type suffix).

Lastly, we will project all the needed columns including key value returned by the first OPENJSON function (L) which will provide us with the SalesOrderLine, which by convention starts at 1 for each order at AdventureWorks, so needs to be adjusted from its zero-based value.

Let's see the result of all this transformation work that the view now contains by:

Paste the following SQL into the query pane.

```
SELECT TOP(10) * FROM SalesOrderDetails
```

The screenshot shows the Microsoft Azure Synapse Analytics query editor interface. The top navigation bar includes 'Microsoft Azure', 'Synapse Analytics', 'synapselinkadventureworks', 'Search', and various toolbar icons. The main area has a query editor with the following content:

```
Synapse live Validate all Publish all
Cosmos DR SQL Script
Run Undo Publish Query plan Connect to Built-in Use database SynapseLinkDB
1 SELECT TOP(10) * FROM SalesOrderDetails
```

Below the editor, the results tab is selected, showing a table with the following data:

| SalesorderId | SalesOrderLine | SKUCode | SKUName | Price | Quantity |
|---------------------------------|----------------|------------|----------------------------------|-----------|----------|
| F8BC4659-6AD8-41FE-BB83-6B3... | 1 | RA-H123 | Hitch Rack - 4-Bike | 120.0000 | 1 |
| E32722EC-3BEF-4378-9291-E5AF... | 1 | HB-M763 | ML Mountain Handlebars | 37.1520 | 1 |
| E32722EC-3BEF-4378-9291-E5AF... | 2 | SE-M940 | HL Mountain Seat/Saddle | 31.5840 | 1 |
| E32722EC-3BEF-4378-9291-E5AF... | 3 | FR-M63S-40 | ML Mountain Frame-W - Silver, 40 | 218.4540 | 5 |
| E32722EC-3BEF-4378-9291-E5AF... | 4 | SH-W890-L | Women's Mountain Shorts, L | 41.9940 | 8 |
| E32722EC-3BEF-4378-9291-E5AF... | 5 | BK-M685-38 | Mountainr-200 Silver, 38 | 1391.9940 | 1 |
| E32722EC-3BEF-4378-9291-E5AF... | 6 | FR-M21S-52 | LL Mountain Frame - Silver, 52 | 158.4300 | 1 |
| E32722EC-3BEF-4378-9291-E5AF... | 7 | CS-4759 | LL Crankset | 105.2940 | 4 |
| E32722EC-3BEF-4378-9291-E5AF... | 8 | CS-9183 | HL Crankset | 242.9940 | 1 |
| E32722EC-3BEF-4378-9291-E5AF... | 9 | FR-M21B-42 | LL Mountain Frame - Black, 42 | 149.8740 | 2 |

At the bottom of the results pane, a message indicates: "00:00:13 Query executed successfully."

Click **run**.

As you can see, we have now extracted the order details information from within the sales order details array including its crucial revenue and unit sales data.

We are now able to create the statistical information needed to answer the questions we set to resolve.

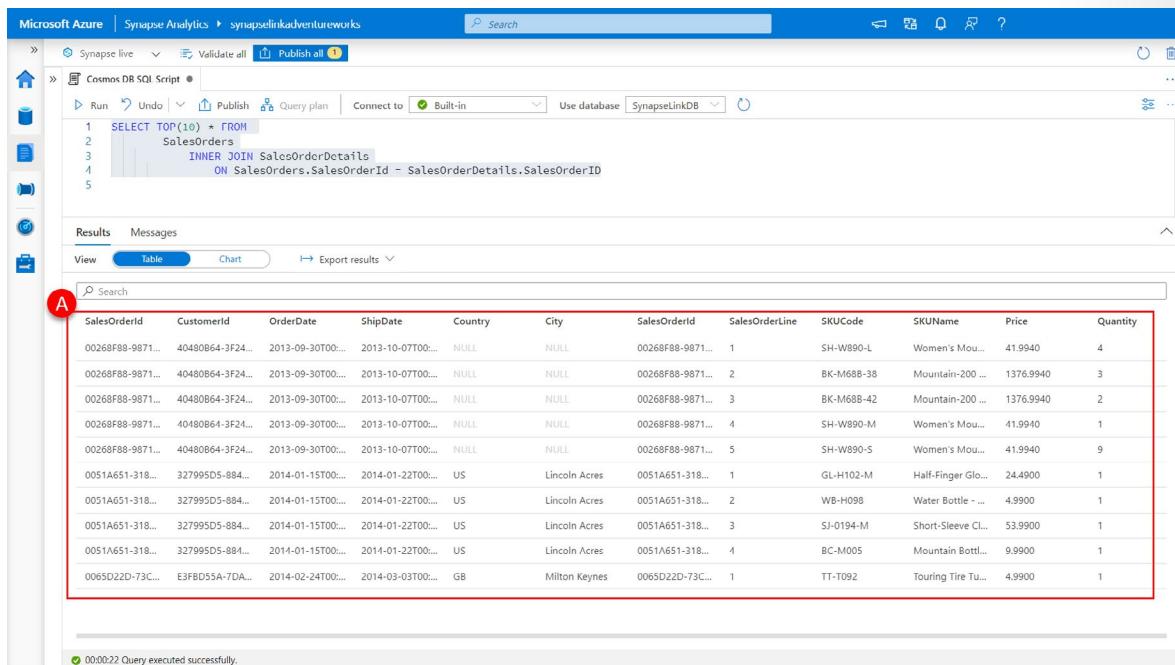
Work with windowing functions and other aggregations

Adventure Works wanted to be able to understand how the sales order volume and revenue is distributed by city for those customers where they have address details. In the previous units, we prepared a SalesOrderView that contains a row for every customer sales order with the country and city information for that customer where that information was available and a SalesOrderDetailsView that contains a row for every sale order line with information on the price and quantity and details of the product sold.

If we join the SalesOrders and SalesOrderDetails views, we have created in the previous units using the SalesOrderId of both we will get a **result set (A)** where single row has both the dimensions across which we want to summarize this data along with the values that we wish to measure.

Paste the following SQL into the query pane.

```
SELECT TOP(10) * FROM
    SalesOrders
    INNER JOIN SalesOrderDetails
        ON SalesOrders.SalesOrderId = SalesOrderDetails.SalesOrderID
```



The screenshot shows the Microsoft Azure Synapse Analytics workspace interface. The top navigation bar includes 'Microsoft Azure', 'Synapse Analytics', and 'synapsesqlinkadventureworks'. Below the navigation is a toolbar with icons for Run, Undo, Publish, Query plan, Connect to, Use database, and a refresh button. The main area is titled 'Cosmos DB SQL Script' with a dropdown menu. The script pane contains the following SQL:

```
1 SELECT TOP(10) * FROM
2     SalesOrders
3     INNER JOIN SalesOrderDetails
4         ON SalesOrders.SalesOrderId = SalesOrderDetails.SalesOrderID
5
```

The results pane is active, showing a table with the following data:

| SalesOrderId | CustomerID | OrderDate | ShipDate | Country | City | SalesOrderId | SalesOrderLine | SKUCode | SKUName | Price | Quantity |
|------------------|------------------|-------------------|-------------------|---------|---------------|------------------|----------------|------------|--------------------|-----------|----------|
| 00268F88-9871... | 40480B64-3F24... | 2013-09-30T00:... | 2013-10-07T00:... | NULL | NULL | 00268F88-9871... | 1 | SH-W890-L | Women's Mou... | 41.9940 | 4 |
| 00268F88-9871... | 40480B64-3F24... | 2013-09-30T00:... | 2013-10-07T00:... | NULL | NULL | 00268F88-9871... | 2 | BK-M68B-38 | Mountainr-200 ... | 1376.9940 | 3 |
| 00268F88-9871... | 40480B64-3F24... | 2013-09-30T00:... | 2013-10-07T00:... | NULL | NULL | 00268F88-9871... | 3 | BK-M68B-42 | Mountainr-200 ... | 1376.9940 | 2 |
| 00268F88-9871... | 40480B64-3F24... | 2013-09-30T00:... | 2013-10-07T00:... | NULL | NULL | 00268F88-9871... | 4 | SH-W890-M | Women's Mou... | 41.9940 | 1 |
| 00268F88-9871... | 40480B64-3F24... | 2013-09-30T00:... | 2013-10-07T00:... | NULL | NULL | 00268F88-9871... | 5 | SH-W890-S | Women's Mou... | 41.9940 | 9 |
| 0051A651-318... | 327995D5-884... | 2014-01-15T00:... | 2014-01-22T00:... | US | Lincoln Acres | 0051A651-318... | 1 | GL-H102-M | Half-Finger Glo... | 24.4900 | 1 |
| 0051A651-318... | 327995D5-884... | 2014-01-15T00:... | 2014-01-22T00:... | US | Lincoln Acres | 0051A651-318... | 2 | WB-I098 | Water Bottle - ... | 4.9900 | 1 |
| 0051A651-318... | 327995D5-884... | 2014-01-15T00:... | 2014-01-22T00:... | US | Lincoln Acres | 0051A651-318... | 3 | SJ-0194-M | Short-Sleeve Cl... | 53.9900 | 1 |
| 0051A651-318... | 327995D5-884... | 2014-01-15T00:... | 2014-01-22T00:... | US | Lincoln Acres | 0051A651-318... | 4 | BC-M005 | Mountain Bottl... | 9.9900 | 1 |
| 0065D22D-73C... | E5FB055A-7DA... | 2014-02-24T00:... | 2014-03-03T00:... | GB | Milton Keynes | 0065D22D-73C... | 1 | TT-T092 | Touring Tire Tu... | 4.9900 | 1 |

A red circle labeled 'A' is positioned over the results table. At the bottom of the results pane, a message says '00:00:22 Query executed successfully.'

Click **run**.

We can now create a single view that will provide us with Country and City level statistics we are after by:

Paste the following SQL into the query pane.

```
CREATE VIEW SalesOrderStats
AS
SELECT
    o.Country, o.City,
    COUNT(DISTINCT o.CustomerId) Total_Customers,
    COUNT(DISTINCT d.SalesOrderId) Total_Orders,
```

```

        COUNT(d.SalesOrderId) Total_OrderLines,
        SUM(d.Quantity*d.Price) AS Total_Revenue,
        dense_rank() OVER (ORDER BY SUM(d.Quantity*d.Price) DESC) as Rank_Revenue,
        dense_rank() OVER (ORDER BY COUNT(DISTINCT d.SalesOrderId) DESC) as Rank_Orders,
        dense_rank() OVER (ORDER BY COUNT(d.SalesOrderId) DESC) as Rank_OrderLines,
        dense_rank() OVER (PARTITION BY o.Country ORDER BY SUM(d.Quantity*d.Price) DESC) as Rank_Revenue_Country
FROM SalesOrders o
INNER JOIN SalesOrderDetails d
    ON o.SalesOrderId = d.SalesOrderId
WHERE Country IS NOT NULL OR City IS NOT NULL
GROUP BY o.Country, o.City
GO

SELECT * FROM SalesOrderStats
GO

```

The query above will create the SalesOrderStats view and output the results of the view created

```

Microsoft Azure | Synapse Analytics > synapselinkadventureworks
Search
» Synapse live | Validate all | Publish all 1
» Cosmos DB SQL Script | Run | Undo | Publish | Query plan | Connect to | Built-in | Use database | SynapseLinkDB | ...
... ...
CREATE VIEW SalesOrderStats
AS
SELECT
    o.Country, o.City,
    COUNT(DISTINCT o.CustomerId) Total_Customers,
    COUNT(DISTINCT d.SalesOrderId) Total_Orders,
    COUNT(d.SalesOrderId) Total_OrderLines,
    SUM(d.Quantity*d.Price) AS Total_Revenue,
    dense_rank() OVER (ORDER BY SUM(d.Quantity*d.Price) DESC) as Rank_Revenue,
    dense_rank() OVER (ORDER BY COUNT(DISTINCT d.SalesOrderId) DESC) as Rank_Orders,
    dense_rank() OVER (ORDER BY COUNT(d.SalesOrderId) DESC) as Rank_OrderLines,
    dense_rank() OVER (PARTITION BY o.Country ORDER BY SUM(d.Quantity*d.Price) DESC) as Rank_Revenue_Country
FROM SalesOrders o
INNER JOIN SalesOrderDetails d
    ON o.SalesOrderId = d.SalesOrderId
WHERE Country IS NOT NULL OR City IS NOT NULL
GROUP BY o.Country, o.City
GO

SELECT * FROM SalesOrderStats
GO

```

Results

| Country | City | Total_Customers | Total_Orders | Total_OrderLines | Total_Revenue | Rank_Revenue | Rank_Orders | Rank_OrderLines | Rank_Revenue_Country |
|---------|-------------|-----------------|--------------|------------------|---------------|--------------|-------------|-----------------|----------------------|
| AU | Wollongong | 105 | 202 | 411 | 338913.4665 | 3 | 23 | 28 | 1 |
| AU | Warrnambool | 105 | 203 | 416 | 327036.3682 | 4 | 22 | 27 | 2 |
| AU | Bendigo | 104 | 201 | 396 | 314568.7193 | 5 | 24 | 33 | 3 |

00:00:21 Query executed successfully.

Click **run**.

The query captured in this view answers the many parts of the questions being asked through traditional aggregation, because we are mostly interested in understanding the number (COUNT) or total (SUM) of values a GROUP BY clause that covers both **Country and City (B)** can answer most of the questions with absolute values for the **total number of customers, orders and order lines and the sum of revenue by City (C)**.

To answer the ranking part of the question, we use window functions. In essence, a window function calculates a result for every row of a table based on a group of rows, called the frame. Every row can have

a unique frame associated with it for that window function allowing you to concisely express and solve ranking, analytic, and aggregation problems in powerful yet simple a manner no other approach does.

Here we **use the dense_rank() function to calculate the rank of each city by revenue, number of orders and total order lines (D)**.

As well as the rank of each city within each country, by partitioning the dense_rank() window function by the country.

Knowledge check

Question 1

Once Azure Synapse Link is configured on Cosmos DB, what is the first step to perform to use Azure Synapse Analytics serverless SQL pools to query the Azure Cosmos DB data?

- Use the OPENROWSET function
- Create a database
- Use a SELECT clause.

Question 2

What function provides a rowset view over a JSON document?

- OPENROWSET.
- OPENJSON.
- WITH.

Summary

In this module, you have learned how to use the Azure Synapse Analytics serverless SQL pools to query the Azure Cosmos DB data made available by Azure Synapse Link.

Now you have completed this module, you'll be able to:

- Write basic queries against a single container
- Perform cross dataset queries in Cosmos DB
- Work with windowing function
- Perform complex queries with JSON data
- Visualize Azure Cosmos DB data in Power BI

Module Lab information

Lab 9 - Support Hybrid Transactional Analytical Processing (HTAP) with Azure Synapse Link

- **Estimated Time:** 50 - 60 minutes
- **Lab files:** The files for this lab are located in the *Instructions\Labs\12* folder.

Lab overview

In this lab, students will learn how Azure Synapse Link enables seamless connectivity of an Azure Cosmos DB account to a Synapse workspace. The student will understand how to enable and configure Synapse link, then how to query the Azure Cosmos DB analytical store using Apache Spark and SQL Serverless.

Lab objectives

After completing this lab, you will be able to:

- Configure Azure Synapse Link with Azure Cosmos DB
- Query Azure Cosmos DB with Apache Spark for Synapse Analytics
- Query Azure Cosmos DB with serverless SQL pool for Azure Synapse Analytics

Module summary

module-summary

In this module, students have learned how to perform operational analytics against Azure Cosmos DB using the Azure Synapse Link feature within Azure Synapse Analytics.

Learning objectives

In this module, you have learned to:

- Design hybrid transactional and analytical processing using Azure Synapse Analytics
- Configure Azure Synapse Link with Azure Cosmos DB
- Query Azure Cosmos DB with Apache Spark for Azure Synapse Analytics
- Query Azure Cosmos DB with SQL Serverless for Azure Synapse Analytics

Post Course Review

After the course, consider visiting the website that explores a **HTAP with Cosmos DB and Synapse Link⁶** pattern and the associated documentation that goes into more depth about this architecture.

Important

Remember

Should you be working in your own subscription while running through this content, it is best practice at the end of a project to identify whether or not you still need the resources you created. Resources left running can cost you money.

You can delete resources one by one, or just delete the resource group to get rid of the entire set.

⁶ <https://docs.microsoft.com/en-us/azure/cosmos-db/synapse-link#:~:text=Azure%20Synapse%20Link%20for%20Azure%20Cosmos%20DB%20is,between%20Azure%20Cosmos%20DB%20and%20Azure%20Synapse%20Analytics>

Answers

Question 1

What is the name of the application architecture that enables near real-time querying to provide insights?

- OLTP.
- HTAP.
- OLAP.

Explanation

Correct. HTAP stands for Hybrid Transactional and Analytical Processing that enable you to gain insights from operational systems without impacting the performance of the operational system.

Question 2

Within Azure Synapse Link for Azure Cosmos DB, which Column-oriented store optimized for queries?

- Transactional store.
- Cosmos DB store.
- Analytical store.

Explanation

Correct. An analytical store is a data store optimized for analytical queries.

Question 3

How can you manage the lifecycle of data and define how long it will be retained for in an analytical store?

- Configure the purge duration in a container
- Configure the default Time to Live (TTL) property for records stored.
- Configure the deletion duration for records in the transactional store.

Explanation

Correct. Configuring the default Time to Live (TTL) property for records stored in an analytical store can manage the lifecycle of data and define how long it will be retained for.

Question 1

Where do you enable Azure Synapse Link for Azure Cosmos DB?

- In Azure Synapse Analytics.
- In Azure Synapse Link.
- In Azure Cosmos DB.

Explanation

Correct. When you enable Azure Synapse Link for Azure Cosmos DB it must be done in Azure Cosmos DB.

Question 2

How do you disable Azure Synapse Link for Azure Cosmos DB?

- Delete the Azure Cosmos DB container.
- Delete the Azure Cosmos DB account.
- Set the Azure Synapse Link option to disable on the Azure Cosmos DB container.

Explanation

Correct. Deleting the Azure Cosmos DB account with disable and remove Azure Synapse Link.

Question 1

When you want to switch to SparkSQL in a notebook, what is the first command to type?

- %%sparksql.
- %%spark.
- %%sql.

Explanation

Correct. When you want to switch to SparkSQL in a notebook, type the %%sql command.

Question 2

What SparkSQL method reads data from the analytical store?

- cosmos.olap.
- cosmos.oltp.
- cosmos.db.

Explanation

Correct. Cosmos.olap is the method that connects to the analytical store in Azure Cosmos DB.

Question 1

Once Azure Synapse Link is configured on Cosmos DB, what is the first step to perform to use Azure Synapse Analytics serverless SQL pools to query the Azure Cosmos DB data?

- Use the OPENROWSET function
- Create a database
- Use a SELECT clause.

Explanation

Correct. Before being able to issue any queries using Azure Synapse Analytics serverless SQL pools, you first must create a database.

Question 2

What function provides a rowset view over a JSON document?

- OPENROWSET.
- OPENJSON.
- WITH.

Explanation

Correct. The OPENJSON function provides a rowset view over a JSON document.

Module 10 Real-time Stream Processing with Stream Analytics

Module introduction

module-introduction

In this module, students will learn the concepts of event processing and streaming data and how this applies to Azure Stream Analytics and Azure Event Hubs. You will understand how to set up a stream analytics job to stream data, and learn how to manage and monitor a running job. You will also learn how to enable reliable messaging for big data applications using Azure Event Hubs.

Learning objectives

In this module, you will:

- Work with data streams by using Azure Stream Analytics
- Enable reliable messaging for Big Data applications using Azure Event Hubs
- Ingest data streams with Azure Stream Analytics

Enable reliable messaging for Big Data applications using Azure Event Hubs

Introduction

Big data apps must be able to process increased throughput by scaling out to meet increased transaction volumes.

Suppose you work in the credit card department of a bank. You're part of a team that manages the system responsible for fraud testing to determine whether to approve or decline each transaction. Your system receives a stream of transactions and needs to process them in real time.

The load on your system can spike during weekends and holidays. The system must handle the increased throughput efficiently and accurately. Given the sensitive nature of the transactions, even the slightest error can have a considerable impact.

Azure Event Hubs can receive and process a large number of transactions. It can also be configured to scale dynamically, when required, to handle increased throughput.

In this module, you'll learn how to connect Event Hubs to your app and reliably process large transaction volumes.

Learning objectives

In this module, you will:

- Create an Event Hub using the Azure CLI.
- Configure apps to send or receive messages through an Event Hub.
- Evaluate Event Hub performance using the Azure portal.

Create an Event Hub using the Azure CLI

Your team has decided to use the capabilities of Azure Event Hubs to manage and process the increasing transaction volumes coming through your system.

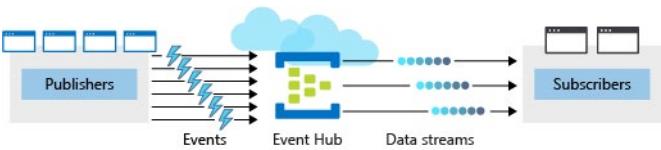
An Event Hub is an Azure resource, so your first step is to create a new hub in Azure, and configure it to meet the specific requirements of your apps.

What is an Azure Event Hub?

Azure **Event Hubs**¹ is a cloud-based, event-processing service that can receive and process millions of events per second. Event Hubs acts as a front door for an event pipeline, to receive incoming data and stores this data until processing resources are available.

An entity that sends data to the Event Hubs is called a *publisher*, and an entity that reads data from the Event Hubs is called a *consumer* or a *subscriber*. Azure Event Hubs sits between these two entities to divide the production (from the publisher) and consumption (to a subscriber) of an event stream. This decoupling helps to manage scenarios where the rate of event production is much higher than the consumption. The following illustration shows the role of an Event Hub.

¹ <https://azure.microsoft.com/services/event-hubs/>



Events

An **event** is a small packet of information (*a datagram*) that contains a notification. Events can be published individually, or in batches, but a single publication (individual or batch) can't exceed 1 MB.

Publishers and subscribers

Event publishers are any app or device that can send out events using either HTTPS or Advanced Message Queuing Protocol (AMQP) 1.0.

For publishers that send data frequently, AMQP has better performance. However, it has a higher initial session overhead, because a persistent bidirectional socket and transport-level security (TLS) or SSL/TLS has to be set up first.

For more intermittent publishing, HTTPS is the better option. Though HTTPS requires additional overhead for each request, there isn't the session initialization overhead.

NOTE: Existing Kafka-based clients, using Apache Kafka 1.0 and newer client versions, can also act as Event Hubs publishers.

Event subscribers are apps that use one of two supported programmatic methods to receive and process events from an Event Hub.

- **EventHubReceiver** - A simple method that provides limited management options.
- **EventProcessorHost** - An efficient method that we'll use later in this module.

Consumer groups

An Event Hub **consumer group** represents a specific view of an Event Hub data stream. By using separate consumer groups, multiple subscriber apps can process an event stream independently, and without affecting other apps. However, the use of many consumer groups isn't a requirement, and for many apps, the single default consumer group is sufficient.

Pricing

There are three pricing tiers for Azure Event Hubs: Basic, Standard, and Dedicated. The tiers differ in terms of supported connections, the number of available Consumer groups, and throughput. When using Azure CLI to create an Event Hubs namespace, if you don't specify a pricing tier, the default of **Standard** (20 Consumer groups, 1000 Brokered connections) is assigned.

Create and configure new Azure Event Hubs

There are two main steps when creating and configuring new Azure Event Hubs. The first step is to define the Event Hubs **namespace**. The second step is to create an Event Hub in that namespace.

Define an Event Hubs namespace

An Event Hubs namespace is a containing entity for managing one or more Event Hubs. Creating an Event Hubs namespace typically involves the following configuration:

Define namespace-level settings

Certain settings such as namespace capacity (configured using **throughput units**), pricing tier, and performance metrics are defined at the namespace level. These settings apply to all the Event Hubs within that namespace. If you don't define these settings, a default value is used: 1 for capacity and *Standard* for pricing tier.

Keep the following aspects in mind:

- You must balance your configuration against your Azure budget expectations.
- You might consider configuring different Event Hubs for different throughput requirements. For example, if you have a sales data app, and you're planning for two Event Hubs, it would make sense to use a separate namespace for each hub.

You'll configure one namespace for high throughput collection of real-time sales data telemetry and one namespace for infrequent event log collection. This way, you only need to configure (and pay for) high throughput capacity on the telemetry hub.

1. Select a unique name for the namespace. The namespace is accessible through this URL: `*namespace*.servicebus.windows.net`
2. Define the following optional properties:
 - Enable Kafka. This option enables Kafka apps to publish events to the Event Hub.
 - Make this namespace zone redundant. Zone-redundancy replicates data across separate data centers with their independent power, networking, and cooling infrastructures.
 - Enable Auto-Inflate and Auto-Inflate Maximum Throughput Units. Auto-Inflate provides an automatic scale-up option by increasing the number of throughput units up to a maximum value. This option is useful to avoid throttling in situations when incoming or outgoing data rates exceed the currently set number of throughput units.

Azure CLI commands to create an Event Hubs namespace

To create a new Event Hubs namespace, use the `az eventhubs namespace` commands. Here's a brief description of the subcommands you'll use in the exercise.

| Command | Description |
|---------------------------------|--|
| <code>create</code> | Create the Event Hubs namespace. |
| <code>authorization-rule</code> | All Event Hubs within the same Event Hubs namespace share common connection credentials. You'll need these credentials when you configure apps to send and receive messages using the Event Hub. This command returns the connection string for your Event Hubs namespace. |

Configure a new Event Hub

After you create the Event Hubs namespace, you can create an Event Hub. When creating a new Event Hub, there are several mandatory parameters.

The following parameters are required to create an Event Hub:

- Event Hub name
 - Event Hub name that is unique within your subscription and:
 - Is between 1 and 50 characters long.
 - Contains only letters, numbers, periods, hyphens, and underscores.
 - Starts and ends with a letter or number.
- **Partition Count** - The number of partitions required in an Event Hub (between 2 and 32). The partition count should be directly related to the expected number of concurrent consumers and can't be changed after the hub has been created. The partition separates the message stream so that consumer or receiver apps only need to read a specific subset of the data stream. If not defined, this value defaults to 4.
- **Message Retention** - The number of days (between 1 and 7) that messages will remain available if the data stream needs to be replayed for any reason. If not defined, this value defaults to 7.

You can also optionally configure an Event Hub to stream data to an Azure Blob storage or Azure Data Lake Store account.

Azure CLI commands to create an Event Hub

To create a new Event Hub with the Azure CLI, you'll run the `az eventhubs eventhub` command set. Here's a brief description of the subcommands we'll be using.

| Command | Description |
|---------------------|---|
| <code>create</code> | Creates the Event Hub in a specified namespace. |
| <code>show</code> | Displays the details of your Event Hub. |

Summary

To deploy Azure Event Hubs, you must configure an Event Hubs namespace, and then configure the Event Hub itself. In the next unit, you'll go through the detailed configuration steps to create a new namespace and Event Hub.

Exercise - Use the Azure CLI to Create an Event Hub

Azure sandbox²

You're now ready to create a new Event Hub. After creating the Event Hub, you'll use the Azure portal to view your new hub.

² <https://docs.microsoft.com/learn/modules/enable-reliable-messaging-for-big-data-apps-using-event-hubs/3-exercise-create-an-event-hub-using-azure-cli>

Create an Event Hubs namespace

Let's create an Event Hubs namespace using Bash shell supported by Azure Cloud shell.

1. First, set default values for the Azure CLI in the Cloud Shell. This will keep you from having to enter these values every time. In particular, let's set the *resource group* and *location*. Enter the following command into the Azure CLI, and feel free to replace the location with one close to you.

The free sandbox allows you to create resources in a subset of the Azure global regions. Select a region from this list when you create resources:

| | |
|--|---|
| <ul style="list-style-type: none">• westus2• southcentralus• centralus• eastus• westeurope | <ul style="list-style-type: none">• southeastasia• japaneast• brazilsouth• australiasoutheast• centralindia |
|--|---|

```
az configure --defaults group=<rgn> [sandbox Resource Group]</rgn> location=westus2
```

2. Create the Event Hubs namespace running the `az eventhubs namespace create` command. Use the following parameters.

| Parameter | Description |
|---|--|
| <code>-name</code> (required) | Enter a 6-50 characters-long unique name for your Event Hubs namespace. The name should contain only letters, numbers, and hyphens. It should start with a letter and end with a letter or number. |
| <code>-resource-group</code> (required) | This will be the pre-created Azure sandbox resource group supplied from the defaults. |
| <code>-l</code> (optional) | Enter the location of your nearest Azure data-center, this will use your default. |
| <code>-sku</code> (optional) | The pricing tier for the namespace [Basic / Standard], defaults to <i>Standard</i> . This determines the connections and consumer thresholds. |

Set the name into an environment variable so we can reuse it.

```
NS_NAME=ehubns-$RANDOM
```

TIP: You can use the Copy button to copy commands to the clipboard. To paste, right-click on a new line in the Cloud Shell window and select Paste or use the Shift+Insert keyboard shortcut ($\square + V$ on macOS).

```
az eventhubs namespace create --name $NS_NAME
```

NOTE: Azure will validate the name you enter, and the CLI returns **Bad Request** if the name exists or is invalid. Try a different name by changing your environment variable and reissuing the command.

3. Fetch the connection string for your Event Hubs namespace running the following command. You'll need this to configure applications to send and receive messages using your Event Hub.

```
az eventhubs namespace authorization-rule keys list \
--name RootManageSharedAccessKey \
--namespace-name $NS_NAME
```

| Parameter | Description |
|--|---|
| <code>--resource-group</code> (required) | This will be the pre-created Azure sandbox resource group supplied from the defaults. |
| <code>--namespace-name</code> (required) | Enter the name of the namespace you created. |

This command returns a JSON block with the connection string for your Event Hubs namespace that you'll use later to configure your publisher and consumer applications. Save the value of the following keys for later use.

- **primaryConnectionString**
- **primaryKey**

Create an Event Hub

Now let's create your new Event Hub.

1. Sign in to the [Azure portal](#)³ using the same account you activated the sandbox with.
2. Create a new Event Hub running the `eventhub create` command. It needs the following parameters.

| Parameter | Description |
|--|----------------------------------|
| <code>--name</code> (required) | Enter a name for your Event Hub. |
| <code>--resource-group</code> (required) | Resource group owner. |
| <code>--namespace-name</code> (required) | Enter the namespace you created. |

Let's define the Event Hub name in an environment variable first.

```
HUB_NAME=hubname-$RANDOM
```

```
az eventhubs eventhub create --name $HUB_NAME --namespace-name $NS_NAME
```

3. View the details of your Event Hub running the `eventhub show` command. It needs the following parameters.

| Parameter | Description |
|--|----------------------------------|
| <code>--resource-group</code> (required) | Resource group owner. |
| <code>--namespace-name</code> (required) | Enter the namespace you created. |
| <code>--name</code> (required) | Name of the Event Hub. |

```
az eventhubs eventhub show --namespace-name $NS_NAME --name $HUB_NAME
```

³ <https://portal.azure.com/learn/docs.microsoft.com>

View the Event Hub in the Azure portal

Next, let's see what this looks like in the Azure portal.

1. Find your Event Hubs namespace using the Search bar at the top of portal.
2. Select your namespace to open it.
3. Select **Event Hubs namespace** under the **ENTITIES** section.
4. Select **Event Hubs**.

Your Event Hub appears with a status of **Active**, and default values for **Message Retention (7)** and **Partition Count** of (4).

The screenshot displays the Azure portal interface for managing Event Hubs. On the left, a sidebar lists various entities: csad73c53def991x4b7dxad7, ehntest78 (selected and highlighted with a red box), and ehstoretest78. Below this, sections include SETTINGS (Shared access policies, Scale, Geo-Recovery, Virtual networks and IP filters, Locks, Automation script) and ENTITIES (Event Hubs, selected and highlighted with a red box). On the right, a larger panel titled 'Event Hub' shows a table with one row: NAME (ehtest78) and STATUS (Active). The entire interface is framed by a light blue border.

Summary

You've now created a new Event Hub, and you've got all the necessary information ready to configure your publisher and consumer applications.

Configure applications to send or receive messages through an Event Hub

After you've created and configured your Event Hub, you'll need to configure applications to send and receive event data streams.

For example, a payment processing solution will use some form of sender application to collect customer's credit card data and a receiver application to verify that the credit card is valid.

Although there are differences in how a Java application is configured, compared to a .NET application, the principles are the same for enabling applications to connect to an Event Hub, and to successfully send or receive messages.

What are the minimum Event Hub application requirements?

To configure an application to send messages to an Event Hub, you must provide the following information, so that the application can create connection credentials:

- Event Hub namespace name
- Event Hub name
- Shared access policy name
- Primary shared access key

To configure an application to receive messages from an Event Hub, provide the following information, so that the application can create connection credentials:

- Event Hub namespace name
- Event Hub name
- Shared access policy name
- Primary shared access key
- Storage account name
- Storage account connection string
- Storage account container name

If you have a receiver application that stores messages in Azure Blob Storage, you'll also need to configure a storage account.

Azure CLI commands to create a general-purpose standard storage account

The Azure CLI provides a set of commands you can use to create and manage a storage account. We'll work with them in the next unit, but here's a basic synopsis of the commands.

TIP: There are several MS Learn modules that cover storage accounts, starting in the module **Introduction to Azure Storage**.

| Command | Description |
|------------------------|--|
| storage account create | Create a general-purpose V2 Storage account. |

| Command | Description |
|--|--|
| storage account key list | Retrieve the storage account key. |
| storage account show-connection-string | Retrieve the connection string for an Azure Storage account. |
| storage container create | Creates a new container in a storage account. |

Shell command to clone an application GitHub repository

Git is a collaboration tool that uses a distributed version control model, and is designed for collaborative working on software and documentation projects. Git clients are available for multiple platforms, including Windows, and the Git command line is included in the Azure Bash cloud shell. GitHub is a web-based hosting service for Git repositories.

If you have an application that is hosted as a project in GitHub, you can make a local copy of the project, by cloning its repository using the **git clone** command.

Edit files in the Cloud Shell

You can use one of the built-in editors in the Cloud Shell to modify all the files that make up the application and add your Event Hub namespace, Event Hub name, shared access policy name, and primary key.

The Cloud Shell supports **nano**, **vim**, **emacs**, and the **Cloud Shell editor (code)**. Just type the name of the editor you want and it will launch in the environment. We'll use the Cloud Shell editor (**code**) editor in the next unit.

Summary

Sender and receiver applications must be configured with specific information about the Event Hub environment. You create a storage account if your receiver application stores messages in Blob Storage. If your application is hosted on GitHub, you have to clone it to your local directory. Text editors, such as **nano** are used to add your namespace to the application.

Exercise - Configure applications to send or receive messages through an Event Hub

Azure sandbox⁴

You're now ready to configure your publisher and consumer applications for your Event Hub.

In this unit, you'll configure these applications to send or receive messages through your Event Hub. These applications are stored in a GitHub repository.

You'll configure two separate applications; one acts as the message sender (**SimpleSend**), the other as the message receiver (**EventProcessorSample**). These are Java applications, which enable you to do everything within the browser. However, the same configuration is needed for any platform, such as .NET.

⁴ <https://docs.microsoft.com/learn/modules/enable-reliable-messaging-for-big-data-apps-using-event-hubs/5-exercise-configure-applications-to-send-or-receive-messages-through-an-event-hub>

Create a general-purpose, standard storage account

The Java receiver application, that you'll configure in this unit, stores messages in Azure Blob Storage. Blob Storage requires a storage account.

1. In the Cloud Shell, create a storage account (general-purpose V2) running the `storage account create` command. Remember we set a default resource group and location, so even though those parameters are normally *required*, we can leave them off.

| Parameter | Description |
|---|---|
| <code>-name</code> (required) | A name for your storage account. |
| <code>-resource-group</code> (required) | The resource group owner. We'll use the pre-created sandbox resource group. |
| <code>-location</code> (optional) | An optional location if you want the storage account in a specific place vs. the resource group location. |

Set the storage account name into a variable. It must be between 3 and 24 characters in length and use numbers and lower-case letters only. It also must be unique within Azure.

```
STORAGE_NAME=storagename$RANDOM
```

Then use this command to create the storage account.

```
az storage account create --name $STORAGE_NAME --sku Standard_RAGRS --encryption-service blob
```

TIP: If the storage account creation fails, change your environment variable and try again.

2. List all the access keys associated with your storage account running the `account keys list` command. It takes your account name and the resource group (which is defaulted).

```
az storage account keys list --account-name $STORAGE_NAME
```

Access keys associated with your storage account are listed. Copy and save the value of **key** for future use. You'll need this key to access your storage account.

3. View the connections string for your storage account running the following command.

```
az storage account show-connection-string -n $STORAGE_NAME
```

This command returns the connection details for the storage account. Copy and save the *value of connectionString*. It should look something like.

```
"DefaultEndpointsProtocol=https;EndpointSuffix=core.windows.net;AccountName=storage_account_name;AccountKey=VZjXuMeuDqjCkT60xx6L5fmtXixYuY2wiPmsrXwY-HIhw0736kSAUAj08XBockRZh7CZwYxuYBPe31hi8XfH1Ww=="
```

4. Create a container called **messages** in your storage account running the following command. Use the **connectionString** you copied in the previous step.

```
az storage container create --name messages --connection-string "<connection string here>"
```

Clone the Event Hubs GitHub repository

Perform the following steps to clone the Event Hubs GitHub repository with `git`. You can execute this right in the Cloud Shell.

1. The source files for the applications that you'll build in this unit are located in a **GitHub repository**⁵.

Run the following commands to make sure that you are in your home directory in Cloud Shell, and then to clone this repository.

```
cd ~  
git clone https://github.com/Azure/azure-event-hubs.git
```

The repository is cloned to your home folder.

Edit SimpleSend.java

You're going to use the built-in Cloud Shell editor. You'll use the editor to modify the `SimpleSend` application, and add your Event Hubs namespace, Event Hub name, shared access policy name, and primary key. The main commands appear at the bottom of the editor window.

You'll need to write out your edits by pressing `Ctrl+O`, and then pressing `Enter` to confirm the output file name. Exit the editor by pressing `Ctrl+X`. Alternatively, the editor has a "..." menu in the top/right corner for all the editing commands.

1. Change to the `SimpleSend` folder.

```
cd ~/azure-event-hubs/samples/Java/Basic/SimpleSend/src/main/java/com/  
microsoft/azure/eventhubs/samples/SimpleSend
```

2. Open the Cloud Shell editor in the current folder. This shows a list of files on the left and an editor space on the right.

```
code .
```

3. Open the `SimpleSend.java` file by selecting it from the file list.

4. In the editor, locate and replace the following strings:

- "Your Event Hubs namespace name" with the name of your Event Hub namespace.
- "Your Event Hub" with the name of your Event Hub.
- "Your policy name" with **RootManageSharedAccessKey**.
- "Your primary SAS key" with the value of the **primaryKey** key for your Event Hub namespace that you saved earlier.

TIP: Unlike the terminal window, the editor can use typical copy/paste keyboard accelerator keys for your OS.

If you've forgotten some of them, you can switch down to the terminal window below the editor and run the `echo` command to list out one of the environment variables. For example:

```
echo $NS_NAME  
echo $HUB_NAME
```

⁵ <https://github.com/Azure/azure-event-hubs>

```
echo $STORAGE_NAME
```

When you create an Event Hubs namespace, a 256-bit SAS key called **RootManageSharedAccessKey** is created that has an associated pair of primary and secondary keys that grant send, listen, and manage rights to the namespace. In the previous unit, you displayed the key running an Azure CLI command, and you can also find this key by opening the **Shared access policies** page for your Event Hubs namespace in the Azure portal.

5. Save **SimpleSend.java** either through the “...” menu, or the accelerator key (Ctrl+S on Windows and Linux, Cmd+S on macOS).
6. Close the editor with the “...” menu, or the accelerator key CTRL+Q.

Use Maven to build SimpleSend.java

You'll now build the Java application running **mvn** commands.

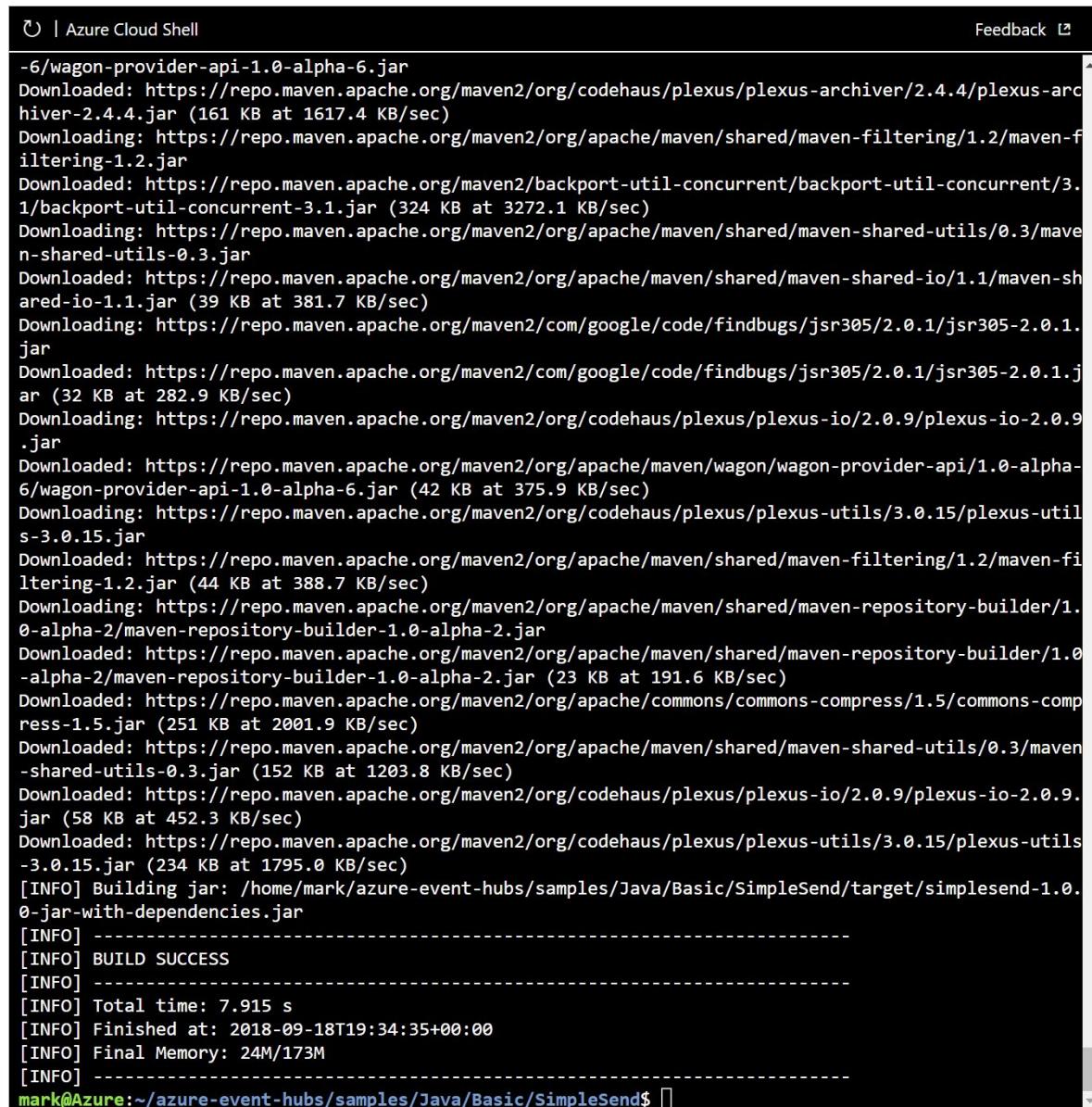
1. Revert to the main **SimpleSend** folder.

```
cd ~/azure-event-hubs/samples/Java/Basic/SimpleSend
```

2. Build the Java SimpleSend application. This ensures that your application uses the connection details for your Event Hub.

```
mvn clean package -DskipTests
```

The build process may take several minutes to complete. Ensure that you see the **[INFO] BUILD SUCCESS** message before continuing.

A screenshot of the Azure Cloud Shell interface. The title bar says "Azure Cloud Shell". On the right, there's a "Feedback" button with a "Send" icon. The main area shows a terminal window with the following text:

```
-6/wagon-provider-api-1.0-alpha-6.jar  
Downloaded: https://repo.maven.apache.org/maven2/org/codehaus/plexus/plexus-archiver/2.4.4/plexus-archiver-2.4.4.jar (161 KB at 1617.4 KB/sec)  
Downloading: https://repo.maven.apache.org/maven2/org/apache/maven/shared/maven-filtering/1.2/maven-filtering-1.2.jar  
Downloaded: https://repo.maven.apache.org/maven2/backport-util-concurrent/backport-util-concurrent/3.1/backport-util-concurrent-3.1.jar (324 KB at 3272.1 KB/sec)  
Downloading: https://repo.maven.apache.org/maven2/org/apache/maven/shared/maven-shared-utils/0.3/maven-shared-utils-0.3.jar  
Downloaded: https://repo.maven.apache.org/maven2/org/apache/maven/shared/maven-shared-io/1.1/maven-shared-io-1.1.jar (39 KB at 381.7 KB/sec)  
Downloading: https://repo.maven.apache.org/maven2/com/google/code/findbugs/jsr305/2.0.1/jsr305-2.0.1.jar  
Downloaded: https://repo.maven.apache.org/maven2/com/google/code/findbugs/jsr305/2.0.1/jsr305-2.0.1.jar (32 KB at 282.9 KB/sec)  
Downloading: https://repo.maven.apache.org/maven2/org/codehaus/plexus/plexus-io/2.0.9/plexus-io-2.0.9.jar  
Downloaded: https://repo.maven.apache.org/maven2/org/apache/maven/wagon/wagon-provider-api/1.0-alpha-6/wagon-provider-api-1.0-alpha-6.jar (42 KB at 375.9 KB/sec)  
Downloading: https://repo.maven.apache.org/maven2/org/codehaus/plexus/plexus-utils/3.0.15/plexus-utils-3.0.15.jar  
Downloaded: https://repo.maven.apache.org/maven2/org/apache/maven/shared/maven-filtering/1.2/maven-filtering-1.2.jar (44 KB at 388.7 KB/sec)  
Downloading: https://repo.maven.apache.org/maven2/org/apache/maven/shared/maven-repository-builder/1.0-alpha-2/maven-repository-builder-1.0-alpha-2.jar  
Downloaded: https://repo.maven.apache.org/maven2/org/apache/maven/shared/maven-repository-builder/1.0-alpha-2/maven-repository-builder-1.0-alpha-2.jar (23 KB at 191.6 KB/sec)  
Downloaded: https://repo.maven.apache.org/maven2/org/apache/commons/commons-compress/1.5/commons-compress-1.5.jar (251 KB at 2001.9 KB/sec)  
Downloaded: https://repo.maven.apache.org/maven2/org/apache/maven/shared/maven-shared-utils/0.3/maven-shared-utils-0.3.jar (152 KB at 1203.8 KB/sec)  
Downloaded: https://repo.maven.apache.org/maven2/org/codehaus/plexus/plexus-io/2.0.9/plexus-io-2.0.9.jar (58 KB at 452.3 KB/sec)  
Downloaded: https://repo.maven.apache.org/maven2/org/codehaus/plexus/plexus-utils/3.0.15/plexus-utils-3.0.15.jar (234 KB at 1795.0 KB/sec)  
[INFO] Building jar: /home/mark/azure-event-hubs/samples/Java/Basic/SimpleSend/target/simplesend-1.0.0-jar-with-dependencies.jar  
[INFO] -----  
[INFO] BUILD SUCCESS  
[INFO] -----  
[INFO] Total time: 7.915 s  
[INFO] Finished at: 2018-09-18T19:34:35+00:00  
[INFO] Final Memory: 24M/173M  
[INFO] -----  
mark@Azure:~/azure-event-hubs/samples/Java/Basic/SimpleSend$
```

Edit EventProcessorSample.java

You'll now configure a **receiver** (also known as **subscribers** or **consumers**) application to ingest data from your Event Hub.

For the receiver application, two classes are available: **EventHubReceiver** and **EventProcessorHost**. **EventProcessorHost** is built on top of **EventHubReceiver**, but provides simpler programmatic interface than **EventHubReceiver**. **EventProcessorHost** can automatically distribute message partitions across multiple instances of **EventProcessorHost** using the same storage account.

In this unit, you'll use the **EventProcessorHost** method. You'll edit the **EventProcessorSample** application to add your Event Hubs namespace, Event Hub name, shared access policy name and primary key, storage account name, connection string, and container name.

1. Change to the **EventProcessorSample** folder running the following command.

```
cd ~/azure-event-hubs/samples/Java/Basic/EventProcessorSample/src/main/java/com/microsoft/azure/eventhubs/samples/eventprocessorsample
```

2. Open the Cloud Shell editor.

```
code .
```

3. Select the **EventProcessorSample.java** file.

4. Locate and replace the following strings in the editor:

- ----ServiceBusNamespaceName---- with the name of your Event Hubs namespace.
- ----EventHubName---- with the name of your Event Hub.
- ----SharedAccessSignatureKeyName---- with **RootManageSharedAccessKey**.
- ----SharedAccessSignatureKey---- with the value of the **primaryKey** key for your Event Hubs namespace that you saved earlier.
- ----AzureStorageConnectionString---- with your storage account connection string that you saved earlier.
- ----StorageContainerName---- with "messages".
- ----HostNamePrefix---- with the name of your storage account.

5. Save **EventProcessorSample.java** either with the "..." menu, or the accelerator key (Ctrl+S on Windows and Linux, Cmd+S on macOS).

6. Close the editor.

Use Maven to build EventProcessorSample.java

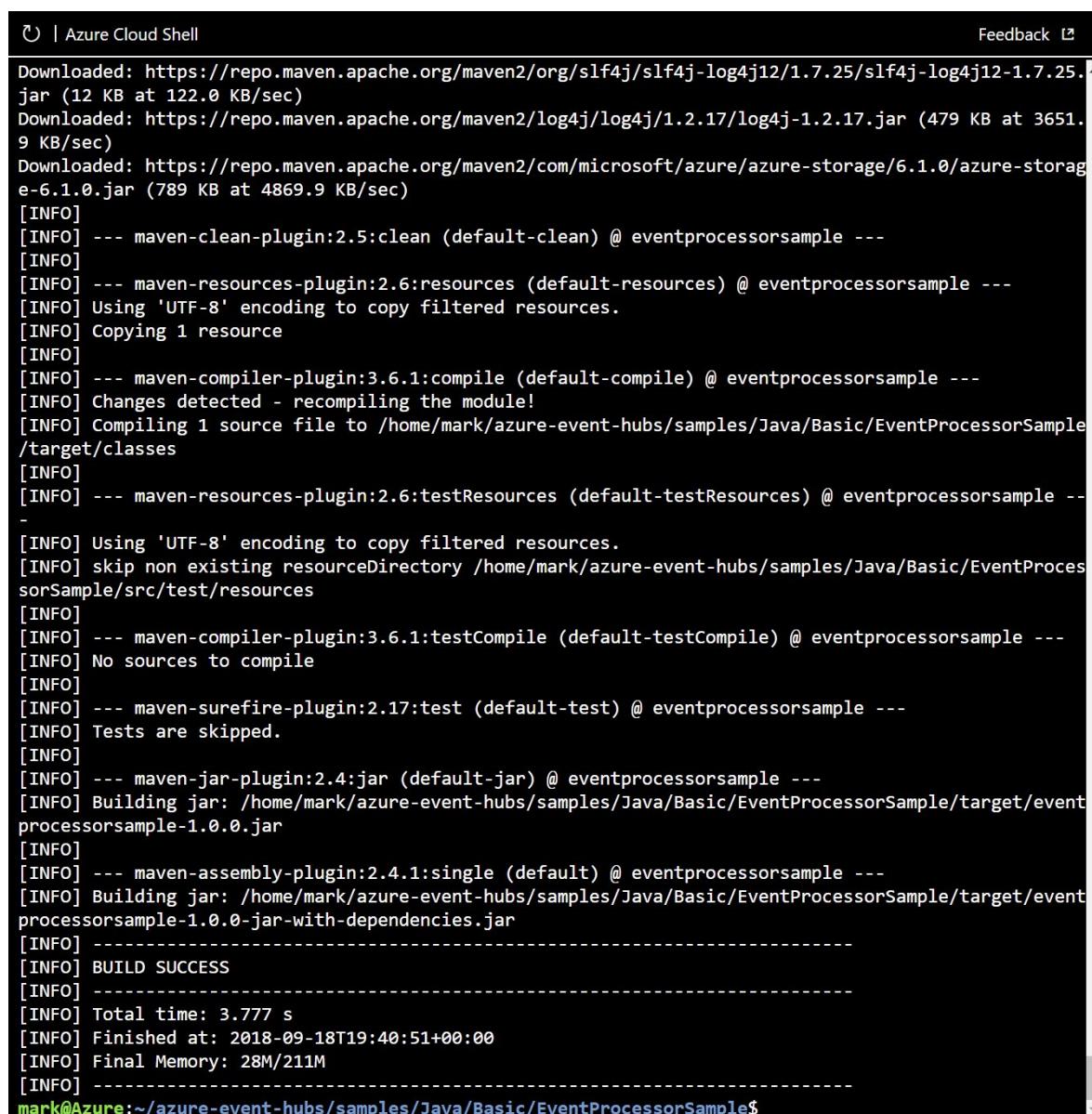
1. Change to the main **EventProcessorSample** folder running the following command.

```
cd ~/azure-event-hubs/samples/Java/Basic/EventProcessorSample
```

2. Build the Java SimpleSend application running the following command. This ensures that your application uses the connection details for your Event Hub.

```
mvn clean package -DskipTests
```

The build process may take several minutes to complete. Ensure that you see a **[INFO] BUILD SUCCESS** message before continuing.



The screenshot shows the Azure Cloud Shell interface with a terminal window. The terminal title is 'Azure Cloud Shell'. The output of a Maven build command is displayed, showing the download of dependencies, the execution of various Maven plugins (clean, resources, compiler, surefire, jar, assembly), and finally a 'BUILD SUCCESS' message. The command at the bottom is 'mark@Azure:~/azure-event-hubs/samples/Java/Basic/EventProcessorSample\$'.

```
Downloaded: https://repo.maven.apache.org/maven2/org/slf4j/slf4j-log4j12/1.7.25/slf4j-log4j12-1.7.25.jar (12 KB at 122.0 KB/sec)
Downloaded: https://repo.maven.apache.org/maven2/log4j/log4j/1.2.17/log4j-1.2.17.jar (479 KB at 3651.9 KB/sec)
Downloaded: https://repo.maven.apache.org/maven2/com/microsoft/azure/azure-storage/6.1.0/azure-storage-6.1.0.jar (789 KB at 4869.9 KB/sec)
[INFO]
[INFO] --- maven-clean-plugin:2.5:clean (default-clean) @ eventprocessorsample ---
[INFO]
[INFO] --- maven-resources-plugin:2.6:resources (default-resources) @ eventprocessorsample ---
[INFO] Using 'UTF-8' encoding to copy filtered resources.
[INFO] Copying 1 resource
[INFO]
[INFO] --- maven-compiler-plugin:3.6.1:compile (default-compile) @ eventprocessorsample ---
[INFO] Changes detected - recompiling the module!
[INFO] Compiling 1 source file to /home/mark/azure-event-hubs/samples/Java/Basic/EventProcessorSample/target/classes
[INFO]
[INFO] --- maven-resources-plugin:2.6:testResources (default-testResources) @ eventprocessorsample --
-
[INFO] Using 'UTF-8' encoding to copy filtered resources.
[INFO] skip non existing resourceDirectory /home/mark/azure-event-hubs/samples/Java/Basic/EventProcessorSample/src/test/resources
[INFO]
[INFO] --- maven-compiler-plugin:3.6.1:testCompile (default-testCompile) @ eventprocessorsample ---
[INFO] No sources to compile
[INFO]
[INFO] --- maven-surefire-plugin:2.17:test (default-test) @ eventprocessorsample ---
[INFO] Tests are skipped.
[INFO]
[INFO] --- maven-jar-plugin:2.4:jar (default-jar) @ eventprocessorsample ---
[INFO] Building jar: /home/mark/azure-event-hubs/samples/Java/Basic/EventProcessorSample/target/eventprocessorsample-1.0.0.jar
[INFO]
[INFO] --- maven-assembly-plugin:2.4.1:single (default) @ eventprocessorsample ---
[INFO] Building jar: /home/mark/azure-event-hubs/samples/Java/Basic/EventProcessorSample/target/eventprocessorsample-1.0.0-jar-with-dependencies.jar
[INFO]
-----
[INFO] BUILD SUCCESS
[INFO]
-----
[INFO] Total time: 3.777 s
[INFO] Finished at: 2018-09-18T19:40:51+00:00
[INFO] Final Memory: 28M/211M
[INFO]
-----
mark@Azure:~/azure-event-hubs/samples/Java/Basic/EventProcessorSample$
```

Start the sender and receiver apps

1. Run Java application from the command line by running the **java** command, and specifying a .jar package. Run the following commands to start the SimpleSend application.

```
cd ~/azure-event-hubs/samples/Java/Basic/SimpleSend
java -jar ./target/simplesend-1.0.0-jar-with-dependencies.jar
```

2. When you see **Send Complete...**, press Enter.

```
jar-with-dependencies.jar
SLF4J: Failed to load class "org.slf4j.impl.StaticLoggerBinder".
SLF4J: Defaulting to no-operation (NOP) logger implementation
SLF4J: See http://www.slf4j.org/codes.html#StaticLoggerBinder for further
details.
```

2018-09-18T19:42:15.146Z: Send Complete...

3. Start the EventProcessorSample application running the following command.

```
cd ~/azure-event-hubs/samples/Java/Basic/EventProcessorSample  
java -jar ./target/eventprocessorsample-1.0.0-jar-with-dependencies.jar
```

4. When messages stop appearing on the console, press Enter or press CTRL+C to end the program.

```
...  
SAMPLE: Partition 0 checkpointing at 1064,19  
SAMPLE (3,1120,20): "Message 80"  
SAMPLE (3,1176,21): "Message 84"  
SAMPLE (3,1232,22): "Message 88"  
SAMPLE (3,1288,23): "Message 92"  
SAMPLE (3,1344,24): "Message 96"  
SAMPLE: Partition 3 checkpointing at 1344,24  
SAMPLE (2,1120,20): "Message 83"  
SAMPLE (2,1176,21): "Message 87"  
SAMPLE (2,1232,22): "Message 91"  
SAMPLE (2,1288,23): "Message 95"  
SAMPLE (2,1344,24): "Message 99"  
SAMPLE: Partition 2 checkpointing at 1344,24  
SAMPLE: Partition 1 batch size was 3 for host mystorageacct2018-46d60a17-  
7060-4b53-b0e0-cca70c970a47  
SAMPLE (0,1120,20): "Message 81"  
SAMPLE (0,1176,21): "Message 85"  
SAMPLE: Partition 0 batch size was 10 for host mystorageacct2018-46d60a17-  
7060-4b53-b0e0-cca70c970a47  
SAMPLE: Partition 0 got event batch  
SAMPLE (0,1232,22): "Message 89"  
SAMPLE (0,1288,23): "Message 93"  
SAMPLE (0,1344,24): "Message 97"  
SAMPLE: Partition 0 checkpointing at 1344,24  
SAMPLE: Partition 3 batch size was 8 for host mystorageacct2018-46d60a17-  
7060-4b53-b0e0-cca70c970a47  
SAMPLE: Partition 2 batch size was 9 for host mystorageacct2018-46d60a17-  
7060-4b53-b0e0-cca70c970a47  
SAMPLE: Partition 0 batch size was 3 for host mystorageacct2018-46d60a17-  
7060-4b53-b0e0-cca70c970a47
```

Summary

You've now configured a sender application ready to send messages to your Event Hub. You've also configured a receiver application ready to receive messages from your Event Hub.

Evaluat the performance of the deployed Event Hub using the Azure portal Copy

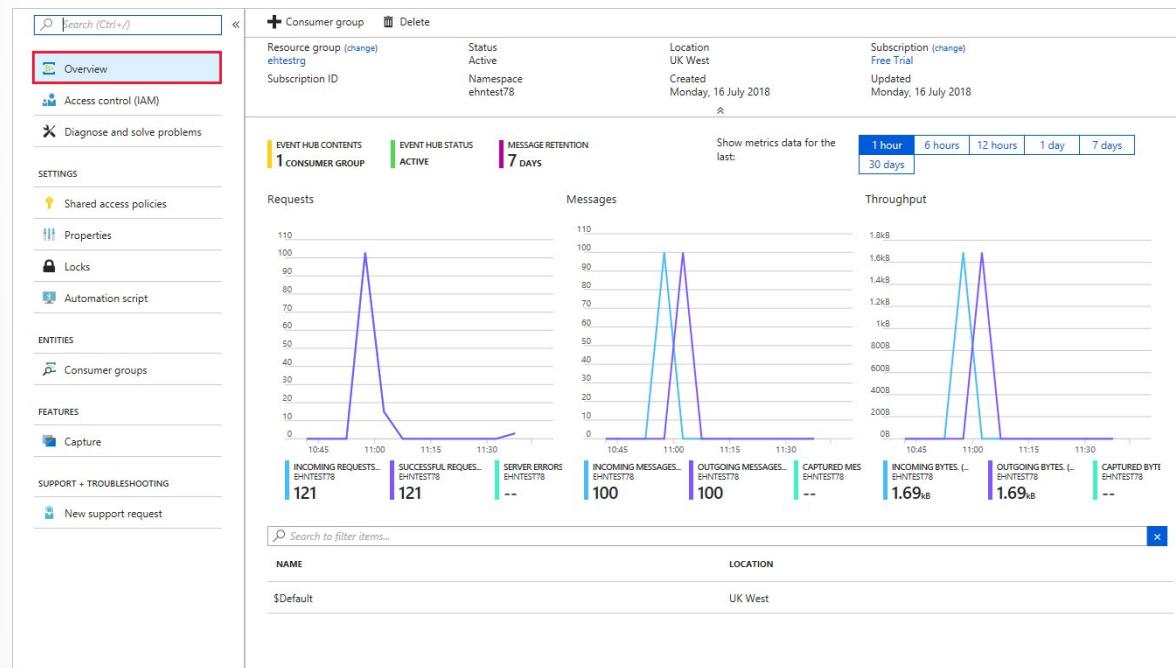
When using Event Hubs, you must monitor your hub to ensure that it's working as expected.

Continuing with the banking example, suppose you've deployed Azure Event Hubs and configured sender and receiver applications. Your applications are ready for testing the payment processing solution. The sender application collects customer's credit card data, and the receiver application verifies the credit card is valid. Because of the sensitive nature of your employer's business, it's essential your payment processing is robust and reliable, even when it's temporarily unavailable.

Evaluate your Event Hub by testing that your Event Hub is processing data as expected. The metrics available in the Event Hubs allow you to ensure that it's working fine.

How do you use the Azure portal to view your Event Hub activity?

The Azure portal > Overview page for your Event Hub shows message counts. These message counts represent the data (events) received and sent by the Event Hub. You can choose the timescale for viewing these events.



How can you test Event Hub resilience?

Azure Event Hubs keeps received messages from your sender application, even when the hub is unavailable. Messages received after the hub becomes unavailable are successfully transmitted to our application as soon as the hub becomes available.

To test this functionality, you can use the Azure portal to disable your Event Hub.

When you re-enable your Event Hub, you can rerun your receiver application, and use Event Hubs metrics for your namespace to check whether all sender messages are successfully transmitted and received.

Other useful metrics available in the Event Hubs include:

- Throttled Requests: The number of throttled requests because the throughput exceeded unit usage.
- ActiveConnections: The number of active connections on a namespace or Event Hub.

- Incoming/Outgoing Bytes: The number of bytes sent to/received from the Event Hubs service over a specified period.

Summary

The Azure portal provides message counts and other metrics that you can use as a health check for your Event Hubs.

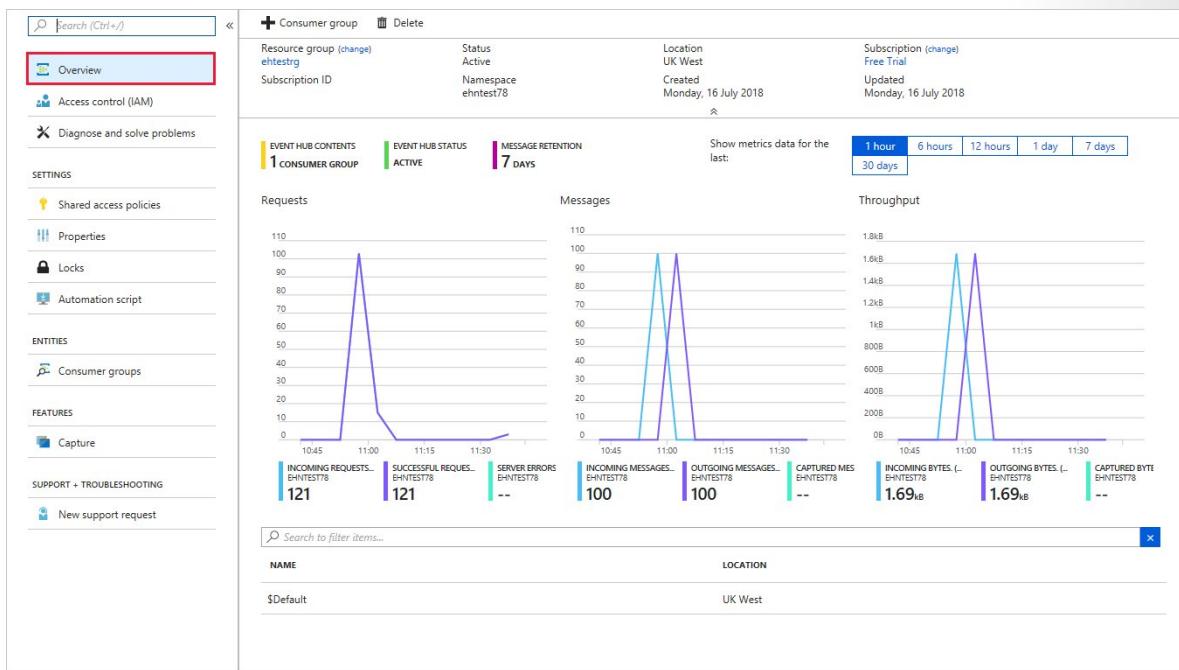
Exercise - Evaluate the performance of the deployed Event Hub using the Azure portal Copy

Azure sandbox⁶

In this unit, you'll use the Azure portal to verify your Event Hub is working according to expectations. You'll also test how Event Hub messaging works when it's temporarily unavailable, and use Event Hubs metrics to check the performance of your Event Hub.

View Event Hub activity

1. Sign into the [Azure portal](#)⁷ using the same account you activated the sandbox with.
2. Find your Event Hub using the Search bar, and open it as we did in the previous exercise.
3. On the Overview page, view the message counts.



4. The SimpleSend and EventProcessorSample applications are configured to send/receive 100 messages. You'll see that the Event Hub has processed 100 messages from the SimpleSend application and has transmitted 100 messages to the EventProcessorSample application.

⁶ <https://docs.microsoft.com/learn/modules/enable-reliable-messaging-for-big-data-apps-using-event-hubs/7-exercise-evaluate-the-performance-of-the-deployed-event-hub-using-the-azure-portal>

⁷ <https://portal.azure.com/learn/docs.microsoft.com>

Test Event Hub resilience

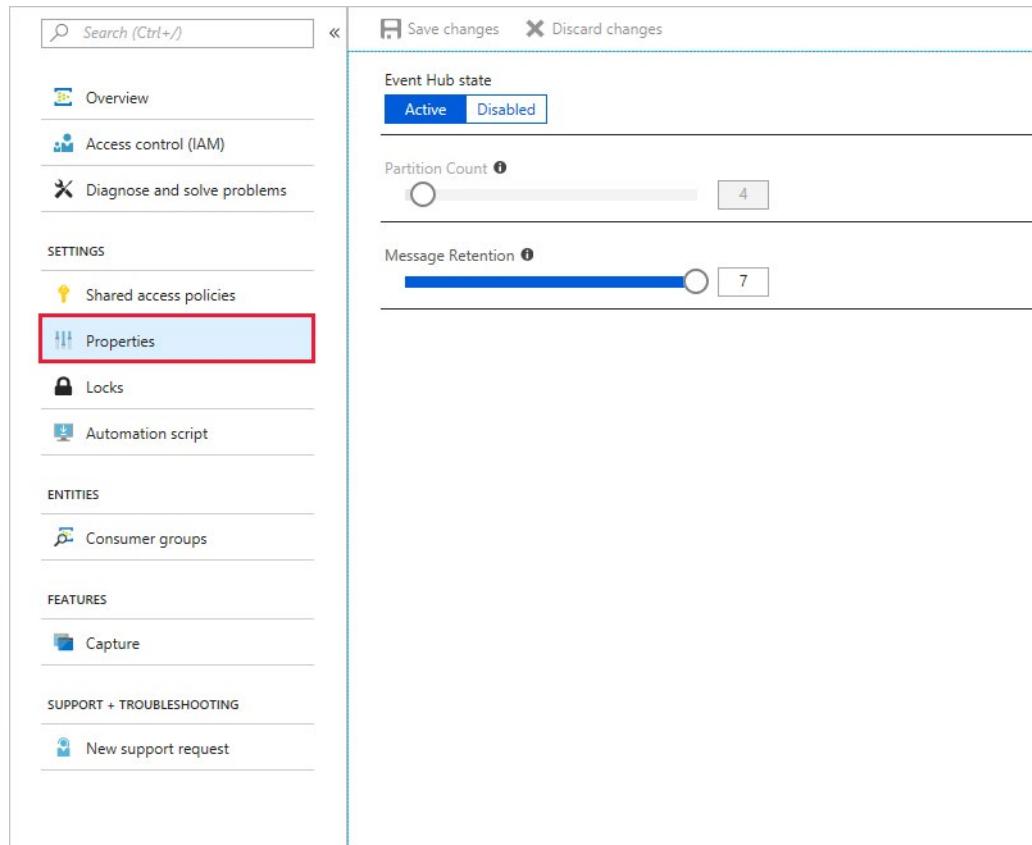
Perform the following steps to see what happens when an application sends messages to an Event Hub while it's temporarily unavailable.

1. Resend messages to the Event Hub using the SimpleSend application. Run the following command.

```
cd ~  
cd azure-event-hubs/samples/Java/Basic/SimpleSend  
java -jar ./target/simplesend-1.0.0-jar-with-dependencies.jar
```

2. When you see **Send Complete**, press Enter.
3. Select your Event Hub in the **Overview** screen - this will show details specific to the Event Hub. You can also get to this screen with the **Event Hubs** entry from the namespace page.
4. Select **Settings > Properties**.

5. Under **EVENT HUB STATUS**, select **Disabled**. Save the changes.

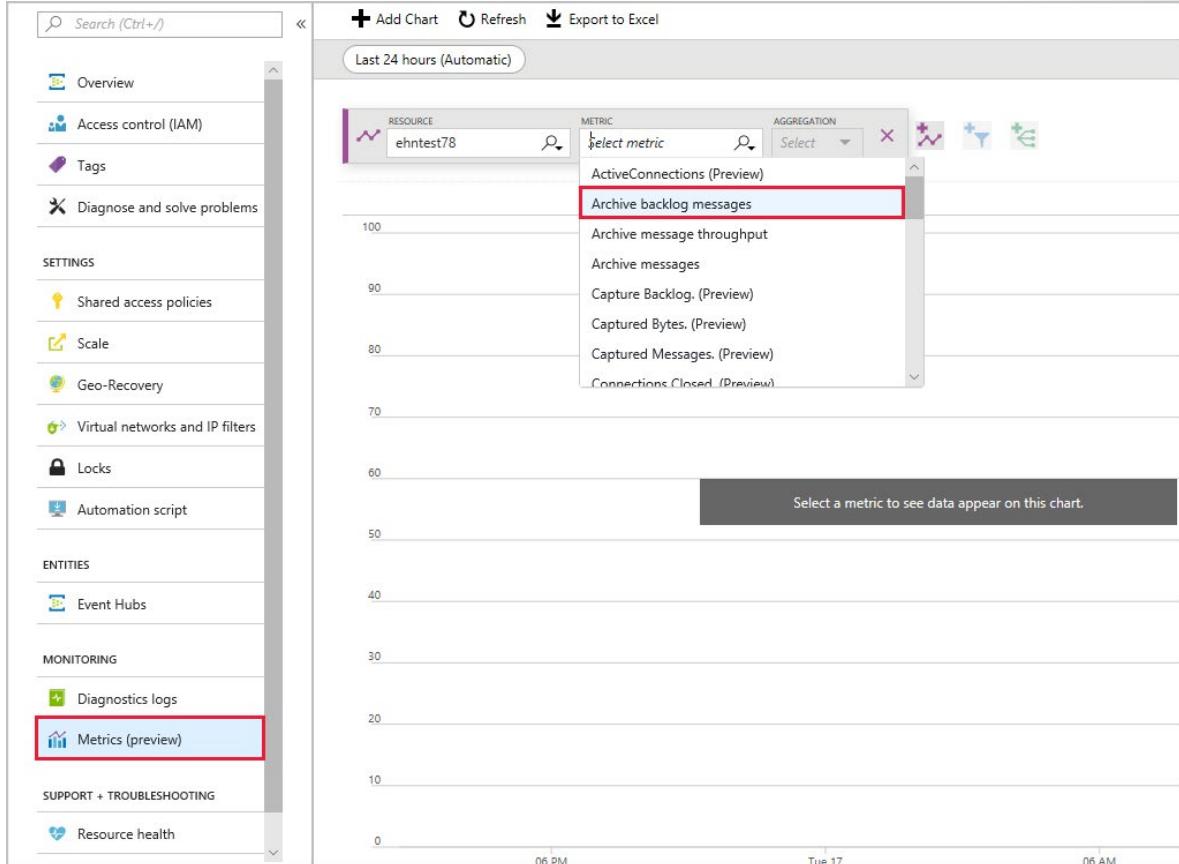


Wait for a minimum of five minutes.

6. Select **Active** under Event Hub state to re-enable your Event Hub, and save your changes.
7. Rerun the EventProcessorSample application to receive messages. Run the following command.

```
cd ~  
cd azure-event-hubs/samples/Java/Basic/EventProcessorSample  
java -jar ./target/eventprocessorsample-1.0.0-jar-with-dependencies.jar
```

8. When messages stop being appearing on the console, press Enter.
 9. Back in the Azure portal, go back to your Event Hub Namespace. If you're still on the Event Hub page, you can use the breadcrumb on the top of the screen to go backwards. Or you can search for the namespace, and select it.
10. Select **Monitoring > Metrics**.



11. From the **Metric** list, select **Incoming Messages**, and then select **Add metric**.
12. From the **Metric** list, select **Outgoing Messages**, and then select **Add metric**.
13. At the top right of the chart, select **Last 24 hours (Automatic)** to change the time period to **Last 30 minutes** to expand the data graph.

You'll see that though the messages were sent before the Event Hub was taken offline for a period, all 100 messages were successfully transmitted.

Summary

In this unit, you used the Event Hubs metrics to test that your Event Hub is successfully processing the sending and receiving messages.

Knowledge check

Question 2

By default, how many partitions will a new Event Hub have?

- 2
- 3
- 4

Question 3

What is the maximum size for a single publication (individual or batch) that is allowed by Azure Event Hub?

- 256 KB
- 1 MB
- 2 MB

Summary

Azure Event Hubs provides big data applications the capability to process large volume of data. It can also scale out during exceptionally high-demand periods, as required. Azure Event Hubs decouples the sending and receiving messages to manage the data processing. This helps eliminate the risk of overwhelming consumer application and data loss because of any unplanned interruptions.

In this module, you've seen how to deploy Azure Event Hubs as part of an event processing solution.

You learned how to:

- Use the Azure CLI commands to create an Event Hubs namespace and an event hub in that namespace.
- Configure sender and receiver applications to send and receive messages through the event hub.
- Use the Azure portal to view your event hub status and performance.

Work with data streams by using Azure Stream Analytics

Introduction

Today, massive amounts of real-time data are generated by connected applications, Internet of Things (IoT) devices and sensors, and various other sources. The proliferation of these streaming data sources has made the ability to consume and make informed decisions from these data in near-real-time an operational necessity for many organizations.

To provide a few examples, online stores analyze real-time clickstream data to provide product recommendations to consumers as they browse the website. Manufacturing facilities utilize telemetry data from IoT sensors to remotely monitor high-value assets. And credit card transactions from point-of-sale systems are scrutinized in real-time to detect and prevent potentially fraudulent activities.

Azure Stream Analytics⁸ seamlessly integrates your real-time application architecture with a streaming analytics engine to transform streaming data into actionable insights. Using Azure Stream Analytics enables powerful, real-time analytics on your data no matter what the volume.

In this module, you will learn how real-time analytics of streaming data works, in principle. You will also discover how you can use Azure Stream Analytics to integrate streaming data into your real-time analytics workflows.

Learning objectives

In this module, you will:

- Understand data streams
- Understand event processing
- Learn about processing events with Azure Stream Analytics

Understand data streams

In the context of analytics, data streams are the data pertaining to the occurrence of specific activities that are emitted by applications, IoT devices or sensors, or other sources known as data producers. These perpetually generated data streams typically contain temporal and additional information about the events. The proliferation of connected applications and devices has led to exponential growth in the number of streaming data sources in recent years.

Data streams are most often used to better understand change over time. For example, an organization may perform sentiment analysis on tweets to see if an advertising campaign results in more positive comments about the company or its products.

Analyzing data streams

In today's world, data streams are ubiquitous. Analyzing a data stream is typically performed to measure how an event's state changes over time or to capture information on an area of interest. The intent being to:

- Continuously analyze data to detect issues and understand or respond to them

⁸ <https://docs.microsoft.com/azure/stream-analytics/stream-analytics-introduction>

- Understand component or system behavior under various conditions to fuel further enhancements of that component or system
 - Trigger specific actions or alerts when certain thresholds are hit

By analyzing data streams and extracting actionable insights, companies can harness latent knowledge to improve efficiencies, further innovation, and respond to irregularities. Examples of use cases that analyze data streams include:

- Stock market trends
- Monitoring data of water pipelines and electrical transmission and distribution systems by utility companies
- Mechanical component health monitoring data in automotive and automobile industries
- Monitoring data from industrial and manufacturing equipment
- Sensor data in transportation, such as traffic management and highway toll lanes
- Patient health monitoring data in the healthcare industry
- Satellite data in the space industry
- Fraud detection in the banking and finance industries
- Sentiment analysis of social media posts

Approaches to data stream processing

There are two approaches to processing data streams: live and on-demand.

The most commonly adopted method for processing data streams is to analyze new data continuously as it arrives from an event producer, such as Azure Event Hubs. This “live” approach requires more processing power to run computations but offers the ability to gain near-real-time insights. Using a service like Azure Stream Analytics, you can execute calculations and aggregations against arriving data using temporal analysis. The results of those queries can be sent to a Power BI dashboard for real-time visualization and analysis.

The diagram below depicts an end-to-end “live” data stream processing solution using Event Hubs to ingest streaming data, Azure Stream Analytics to transform data, and Power BI to visualize and analyze it.

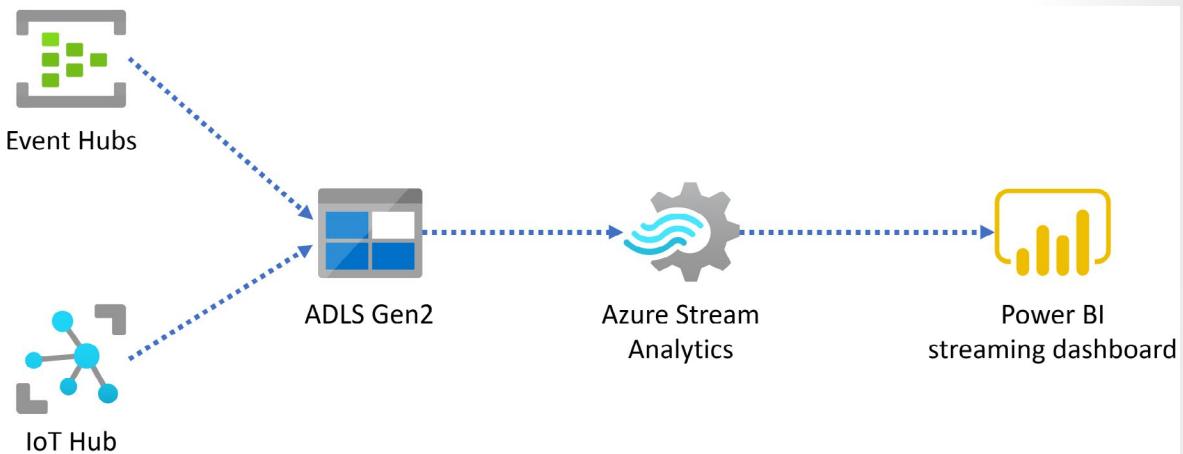


The “on-demand” approach for processing streaming data involves persisting all incoming data in a data store, such as **Azure Data Lake Storage (ADLS) Gen2**⁹. This method allows you to collect streaming data over time and store it as static data. You can then process the *static* data in batches when convenient or during times when compute costs are lower.

The following diagram illustrates an on-demand data stream processing solution. Streaming data from Azure Event Hubs and IoT Hub are written as blobs into Azure Data Lake Storage (ADLS) Gen2. The static

⁹ <https://docs.microsoft.com/azure/storage/blobs/data-lake-storage-introduction>

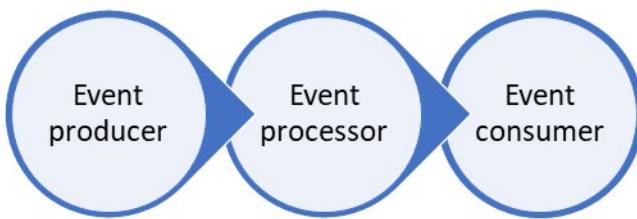
data are then processed using Azure Stream Analytics and output to a Power BI dashboard for visualization and analysis.



Understand event processing

Event processing refers to the consumption and analysis of a continuous data stream to extract latent knowledge and derive actionable insights from the events happening within that stream. Event processing pipelines provide an end-to-end solution for ingesting, transforming, and analyzing data streams and are made up of three distinct components:

- An **event producer**, which generates an event data stream
- An **event processor** responsible for the ingestion and transformation of streaming event data
- An **event consumer** that displays or consumes event data and takes action on it



Event producer

An event producer is an application, connected device or sensor, or any other service that continuously outputs a data stream of events. Events can be any recurring action, such as a heartbeat, a car passing through a toll booth, or an engine sensor reporting temperature values on an automobile. Every event should include temporal information, such as a timestamp, indicating when it occurred.

Azure Event Hubs and IoT Hub are frequently used as event producers in Azure. These services can handle incoming events at a massive scale and produce an event stream to feed into downstream event processing services, such as Azure Stream Analytics.

Event processor

An event processor is an engine designed to consume and transform event data streams. Event processors require the ability to query time segments easily. Performing time-boxed computations or aggregations, such as counting the number of times an event happens during a particular period, is a frequent use case. Depending on the problem space, event processors either process one incoming event at a time, such as a heart rate monitor, or process multiple events simultaneously, such as Azure Stream Analytics processing sensor data from highway tollbooths.

Azure Stream Analytics¹⁰ is the quickest way to get event processing running on Azure. Using Stream Analytics, you can ingest streaming data from Azure Events Hubs or IoT Hub and run real-time analytics queries against the streams.

Stream Analytics supports multiple types of windowing functions for performing temporal computations on data streams, providing a way to aggregate events over various time intervals depending on specific window definitions. It also provides the capability to use Azure Machine Learning functions to make it a robust tool for analyzing data streams.

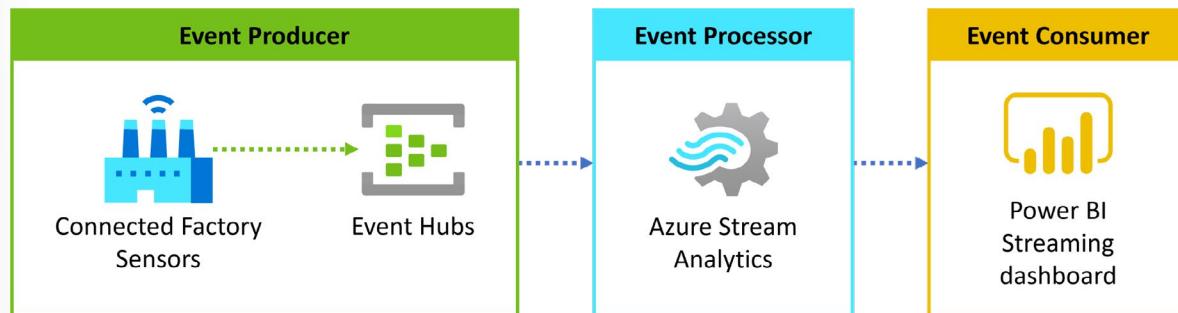
Event consumer

An event consumer is an application that consumes the output of an event processor. Event consumers can be used to visualize data or take a specific action based on the insights, such as generating alerts when specified thresholds are met or sending data to another event processing engine.

For visualizing and analyzing data, **Power BI**¹¹ is the recommended event consumer. It provides a platform for creating complex linked visualizations and analyzing aggregated data in near-real-time. Azure Stream Analytics can output directly to Power BI, allowing dashboards to be updated continuously as data streams are processed.

Building event processing pipelines

Event processing pipelines generally chain together multiple services to create a near-real-time analytics pipeline. The diagram below shows an example event processing pipeline built on Event Hubs, Azure Stream Analytics, and Power BI.



In this pipeline, Event Hubs ingests streaming data from sensors in a connected factory, and together they act as the event producer. The event processor is Azure Stream Analytics, which receives the data from Event Hubs and transforms and aggregates it. Power BI, the event consumer, receives output from Stream Analytics and displays rich visualizations used to analyze the event data.

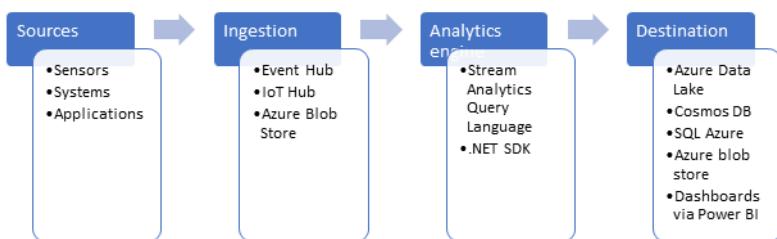
¹⁰ <https://docs.microsoft.com/azure/stream-analytics/stream-analytics-introduction>

¹¹ <https://docs.microsoft.com/power-bi/fundamentals/power-bi-overview>

Process events azure stream analytics

Azure Stream Analytics is a platform-as-a-service (PaaS) event processing engine. It enables the transformation and analysis of large volumes of streaming data arriving from Azure Event Hubs and IoT Hub and static data from Azure storage. Using Stream Analytics, you can write complex time-based queries and aggregations over the data generated by connected sensors, devices, or applications. Stream Analytics processes the data in real-time, enabling powerful insights to drive real-time decision-making. A typical event processing pipeline built on top of Stream Analytics consists of the following four components:

- **Event producer:** Any application, system, or sensor that continuously produces event data of interest. Examples include sensors tracking the flow of water through a utility pipe and an application such as Twitter that generates tweets against a single hashtag.
- **Event ingestion system:** Receives the data from an event producer and passes it to an analytics engine. Azure Event Hubs, Azure IoT Hub, or Azure Blob storage can serve as the ingestion system.
- **Stream analytics engine:** The compute platform that processes, aggregates, and transforms incoming data streams. Azure Stream Analytics provides the Stream Analytics query language (SAQL), a subset of Transact-SQL tailored to perform computations over streaming data. The engine supports windowing functions that are fundamental to stream processing and are implemented by using the SAQL.
- **Event consumer:** A destination of the output from the stream analytics engine. The output can be stored in a data storage platform, such as Azure Data Lake Storage Gen2, Azure Cosmos DB, Azure SQL Database, or Azure Blob storage. Or, you can consume the output in near-real-time using Power BI dashboards.



Operational aspects

Stream Analytics guarantees *exactly once* event processing and *at-least-once* event delivery, so events are never lost. It has built-in recovery capabilities in case the delivery of an event fails. Also, Stream Analytics provides built-in checkpointing to maintain the state of your job and produces repeatable results.

Because Azure Stream Analytics is a PaaS service, it's fully managed and highly reliable. Its built-in integration with various sources and destinations and flexible programmability model enhance programmer productivity. The Stream Analytics engine enables in-memory compute, so it offers superior performance. All these factors contribute to the low total cost of ownership (TCO) of Azure Stream Analytics.

Knowledge check

Question 1

Which of the following technologies typically provide an ingestion point for data streaming in an event processing solution that uses static data as a source?

- Azure IoT Hub
- Azure Blob storage
- Azure Event Hub

Question 2

To consume processed event streaming data in near-real-time to produce dashboards containing rich visualizations, which of the following services should you use?

- Azure Cosmos DB
- Event Hubs
- Power BI

Summary

Azure Stream Analytics is a PaaS service that integrates with your applications and Internet of Things (IoT) to gain insights with streaming data or static data held in a blob store. The process of consuming data streams, analyzing them, and deriving actionable insights out of them is called event processing. It requires an event producer, an event processor, and an event consumer. Azure Stream Analytics provides the event processing aspect to streaming that's fully managed and highly reliable.

In this module, you learned how real-time analytics of streaming data works, in principle. You also discovered how you can use Azure Stream Analytics to integrate streaming data into your real-time analytics workflows.

Learning objectives

In this module, you have:

- Understood data streams
- Understood event processing
- Learned about processing events with Azure Stream Analytics

Transform data by using Azure Stream Analytics

Introduction

In Azure Stream Analytics, jobs are the foundation of real-time analytics. The good news is that setting up jobs is straightforward:

- Set the job inputs to ingest data.
- Define how to process the data.
- Set up a job output to store or display the results.

Let's say that you're demonstrating to the Contoso team how to set up Stream Analytics. You first provide a high-level workflow. Then you show how to create a job input and a job output. Finally, you provide example queries for the analytics. You show the team how to start the job and verify the result.

Learning objectives

In this module you will:

- Explore the Stream Analytics workflow.
- Create a Stream Analytics job.
- Set up a Stream Analytics job input.
- Set up a Stream Analytics job output.
- Write a transformation query.
- Start a Stream Analytics job.

Understand the Streaming Analytics Workflow

In Azure Stream Analytics, a *job* is a unit of execution. A Stream Analytics job pipeline consists of three parts:

- An **input** that provides the source of the data stream.
- A **transformation query** that acts on the input. For example, a transformation query could aggregate the data.
- An **output** that identifies the destination of the transformed data.

The Stream Analytics pipeline provides a transformed data flow from input to output, as the following diagram shows.



Exercise - Create an Azure Stream Analytics job

Let's start by creating a new Azure Stream Analytics job in the Azure portal.

NOTE:This exercise is optional. If you don't have an Azure account or prefer not to do the exercise in your account, you can just read through the instructions. This overview will help you understand the steps to create a Stream Analytics job.

1. Open the **Azure portal**¹² and sign in to your account.
2. Select **More services**, and then select **All services**.
3. In the search box, enter **Stream Analytics**, and select **Stream Analytics jobs** from the results.
4. On the **Stream Analytics jobs** page, select **+ New**.

The screenshot shows the 'Stream Analytics jobs' page in the Azure portal. At the top, there's a header with 'All services >' and the Microsoft logo. Below it is a toolbar with buttons for '+ New', 'Manage view', 'Refresh', 'Export to CSV', 'Open query', 'Assign tags', 'Feedback', and 'Leave preview'. There are also filter options for 'Subscription' (set to 'Learn AIRS - Microsoft Azure Internal Consumption'), 'Resource group' (set to 'all'), 'Location' (set to 'all'), and 'Status'. A search bar says 'Filter for any field...' with the placeholder 'Subscript... == Learn AIRS - Microsoft Azure Internal Consump...'. Below the toolbar, a message says 'Showing 0 to 0 of 0 records.' A large gear icon with a wavy line is centered on the page. Below the gear, a message says 'No stream analytics jobs to display. Try changing your filters if you don't see what you're looking for.' A 'Learn more' link is provided. At the bottom right is a blue button labeled 'Create stream analytics job'.

The **New Stream Analytics job** page appears.

5. Enter a **Job name**, such as **SimpleTransformer**.
6. In the **Resource group** field, select **Create new**, and enter **mslearn-streamanalytics**. Select **OK**.
7. Note the **Location** setting. Ideally, you should create your job in the same location as any storage accounts you use as a source or destination.
8. Ensure **Cloud** is selected for the **Hosting environment**.
9. Set the **Streaming units** to **1** to minimize the cost for this test.
10. Select **Create** to create the new job. Your job will go through validation.

¹² <https://portal.azure.com/>

All services > Stream Analytics jobs >

New Stream Analytics job

This will create a new Stream Analytics job. You will be charged according to Azure Stream Analytics billing model. Learn more.

Job name *
SimpleTransformer.

Subscription *
Learn AIRS - Microsoft Azure Internal Consumption

Resource group *
mslearn-streamanalytics

Create new

Location *
East US

Hosting environment
Cloud Edge

Streaming units (1 to 192)
1

Secure all private data assets needed by this job in my Storage account.

Create

11. After a few moments, select **Refresh** to see your new Stream Analytics job.

| Stream Analytics jobs | | | |
|--|---------|-----|---|
| Subscriptions: 4 of 11 selected – Don't see a subscription? Open Directory + Subscription settings | | | |
| NAME | STATUS | CRE | A |
| SimpleTransformer | Created | 201 | |

Now that we have a Stream Analytics job, we're ready to set up the job to serve a streaming workload. We'll start with the input.

Exercise - Configure the Azure Stream Analytics job input

An Azure Stream Analytics job supports three input types:

| Input type | Use case |
|---------------------------|--|
| Azure Event Hub | Azure Event Hub consumes live streaming data from applications with low latency and high throughput. |
| Azure IoT Hub | Azure IoT Hub consumes live streaming events from IoT devices. This service enables bi-directional communication scenarios where commands can be sent back to IoT devices to trigger specific actions based on analyzing streams they send to the service. |
| Azure Blob Storage | Azure Blob Storage is used as the input source to consume files persisted in blob storage. |

Create the input source

Let's use an Azure Blob store as the input. Recall that Azure Blob Storage has three aspects to it:

- Storage account to provide the globally unique namespace in Azure
- Container which acts like a folder
- Blob itself which is similar to a file in a file system

Let's start by creating an Azure Blob Storage account.

1. Switch back to the [Azure portal](#)¹³.
2. Select **All services** in the top left corner.
3. Enter “**storage**” in the search field, and select **Storage accounts** from the results.
4. Select **+ New** to create a new Azure Storage account.
5. On the **Basics** tab, from the **Resource group** dropdown, select the new **mslearn-streamanalytics** resource group.
6. Set the **Storage account name** to a unique name. Enter the prefix “streams” with your initials or a numeric value. This value has to be unique across all Azure storage accounts, so you might have to try a few combinations to find one that works for you. The portal will place a green checkmark next to the name if it's valid.
7. Check the **Location**. Set it to the same location as the job to avoid having to pay to transfer data between regions.
8. Leave the rest of the fields as default values.

¹³ <https://portal.azure.com/>

Create storage account

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: mslearn-streamanalytics [Create new](#)

INSTANCE DETAILS

The default deployment model is Resource Manager, which supports the latest Azure features. You may choose to deploy using the classic deployment model instead. [Choose classic deployment model](#)

* Storage account name: streamsrsrc123
* Location: East US
Performance: Standard (radio button selected) Premium
Account kind: StorageV2 (general purpose v2)
Replication: Read-access geo-redundant storage (RA-GRS)
Access tier (default): Cool (radio button) Hot (radio button selected)

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

9. Select **Review + create**.

10. After the request has been validated, select **Create** to submit the deployment request.

Wait a few moments for the deployment to complete. After the message "Your deployment is complete" appears, go to the next step.

Connect the input source to the Stream Analytics job

Next, let's connect our Stream Analytics job to our new Blob Storage account.

1. In the Azure portal, select **All services** in the left sidebar.
2. In the search box, enter **Stream Analytics**. Select the **Stream Analytics jobs** from the results.
3. In the list of jobs, select the Stream Analytics job you created earlier (**SimpleTransformer**). The overview page for your job appears.

The screenshot shows the 'SimpleTransformer' Stream Analytics job in the Azure portal. The left sidebar contains navigation links like Overview, Activity log, Access control (IAM), Tags, Diagnose and solve problems, Settings, Properties, Locks, Job topology, Inputs, Functions, Query, Outputs, Configure, Environment, Storage account settings, Scale, Locale, Event ordering, Error policy, Compatibility level, and Managed identity. The main content area has tabs for Overview, Inputs (0), Outputs (0), Monitoring, and Resource utilization. The Overview tab displays job details: Resource group (mslearn-streamanalytics), Status (Created), Location (East US), Subscription (Learn AIRS - Microsoft Azure Internal Consumption), Subscription ID (888ace80-a2c5-41c3-a2ce-367ee5db4225), and system metrics like Send feedback (UserVoice), Created (Monday, February 1, 2021, 12:32:43 PM), Started (not yet), Output watermark (not yet), and Hosting environment (Cloud). A 'Query' section shows a sample T-SQL query:

```
1 SELECT *  
2 INTO [YourOutputAlias]  
3 FROM [YourInputAlias]
```

4. Under **Job topology**, select **Inputs**.
5. Select **Add stream input**, and select **Blob storage/ADLS Gen2** from the dropdown. The **Blob storage/ADLS Gen2** panel appears.
6. Enter **streaminput** in **Input alias** field. This is a friendly name you use to identify the input.
7. Select the **Storage account** you created previously from the dropdown. Recall it starts with **streams-rc**.
8. Select **Create New** for the **Container** field, and give it a unique name, such as **learncontainer**.
9. Select **Connection string** from the dropdown for **Authentication mode**.
10. Enter **input/** for **Path pattern**.
11. Leave the rest of the fields with the current default values.

Blob storage/ADLS Gen2

New input

Input alias *
streaminput ✓

Provide Blob storage/ADLS Gen2 settings manually
 Select Blob storage/ADLS Gen2 from your subscriptions

Subscription
[dropdown menu] ▾

Storage account *
ctostreamrc ▾

Container *
 Create new Use existing
learncontainer ✓

Authentication mode
Connection string ▾

Storage account key

Path pattern ⓘ
input/ ✓

Date format
YYYY/MM/DD ▾

Time format
HH ▾

Partition key ⓘ
[dropdown menu] ✓

Count of input partitions ⓘ
1

Save

12. Select **Save** to associate the input.

Exercise - Configure the Azure Stream Analytics job output

Stream Analytics jobs support various output sinks, such as Azure Blob storage, SQL Database, and Event Hubs. The documentation lists all output types.

In this exercise, we'll use Blob storage as the output sink for our Stream Analytics job.

NOTE: This exercise is optional. If you don't have an Azure account or prefer not to do the exercise in your account, just read through the instructions so you understand the steps involved in creating an output sink for a Stream Analytics job.

These steps are similar to the ones you followed to create the input. Start by creating a second Blob storage account to hold the output.

1. In the **Azure portal**¹⁴, create a new storage account, just like you did in the previous exercise.

| Setting | Value |
|---|---|
| On the Basics tab, under Project details section: | |
| Resource group | Select Create new link; enter mslearn-streamanalytics in the Name box, and select OK . |
| Under Instance details section: | |
| Storage account name | Enter the prefix streamsink , and add a numeric suffix. You might need to try a few combinations to find a unique name in Azure. |
| <i>remaining settings</i> | Leave <i>default</i> . |

2. Select **Review + create**.
3. After the request is validated, select **Create** to run the deployment step.

Wait until a message indicates that the deployment is complete before continuing to the next step.

Connect an output sink to a Stream Analytics job

Next, connect the storage account as the destination for the Stream Analytics job.

1. On the Azure portal **Home** page, select **All services**.
2. In the search box, enter **Stream Analytics**, and select **Stream Analytics jobs** from the results.
3. Select the Stream Analytics job you created.
4. In the left nav bar, under **Job topology**, select **Outputs**.
5. Select **+ Add**, and from the list, select **Blob storage/ADLS Gen2**. The **Blob storage/ADLS Gen2** panel appears.

| Setting | Value |
|--|-------------------------|
| Output alias | Your output alias name. |
| Select storage from your subscriptions | checked |
| Subscription | Your subscription name. |

¹⁴ <https://portal.azure.com/>

| Setting | Value |
|---------------------------|--|
| Storage account | :awsastudxx:, where xx is your initials. |
| Container | Use existing (checked) |
| <i>Remaining settings</i> | Leave <i>default</i> |

6. Select **Save**.

NOTE: If your account doesn't appear in the list, try refreshing it by closing the Azure portal, closing the browser, and then opening the Azure portal again.

7. Under **Container**, select **Create new**. Give the container a unique name, such as **learn-container**.
8. Under **Path pattern**, enter **output/**.
9. Leave the default values in the rest of the fields.
10. Select **Save**.

Exercise - Write an Azure Stream Analytics transformation query

An Azure Stream Analytics query transforms an input data stream and produces an output. Queries are written in a language like SQL that's a subset of the Transact-SQL (T-SQL) language.

In this exercise, we'll transform the input data in a simple way to demonstrate the transformation-query capabilities that Stream Analytics exposes.

NOTE: This exercise is optional. If you don't have an Azure account or prefer not to do the exercise in your account, just read through the instructions so you understand how to set up an input file and change the transformation query in a Stream Analytics job.

Let's imagine you need to pull elements out of some census data. In this case, you want the coordinates of each city in the census data. You'll use a simple JSON file as your input. Run a transformation query to pull coordinates out of the data, and then write the results to a new file in your Blob storage.

Create the sample input file

Start by creating a simple input file named **input.json** on your local computer. The file has these contents.

```
{
    "City" : "Reykjavík",
    "Coordinates" :
    {
        "Latitude": 64,
        "Longitude": 21
    },
    "Census" :
    {
        "Population" : 125000,
        "Municipality" : "Reykjavík",
        "Region" : "Höfuðborgarsvæðið"
    }
}
```

Upload the input file

Next, upload the JSON file to a Blob storage container.

1. Open the [Azure portal](#)¹⁵.
2. Go to your source Blob storage account (**streamsdc**).

TIP: To find resources by name, use the Search field at the top of the Azure portal. To find related resources, use your resource group.

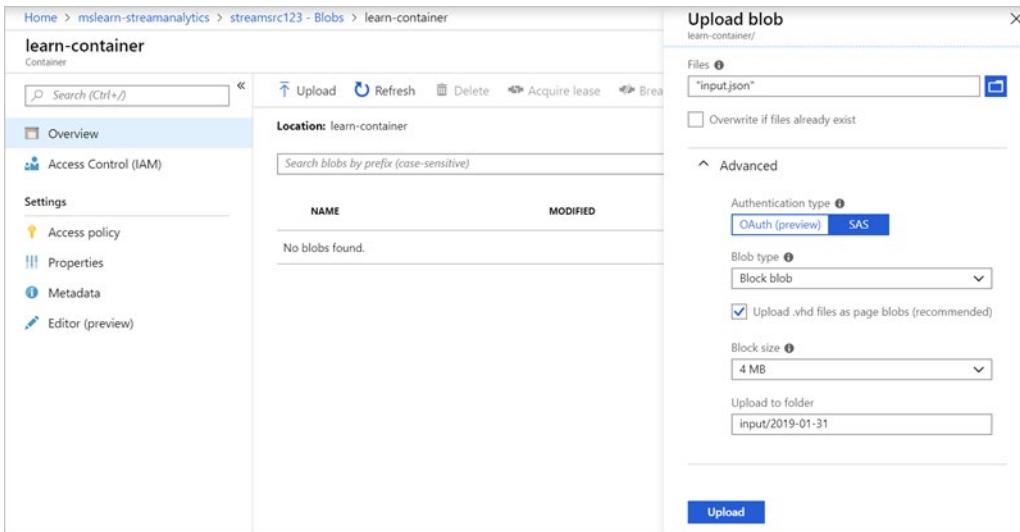
3. Select the **streamsource** storage account you created earlier.
4. Under **Blob service**, select **Containers**.

The screenshot shows the Azure portal interface for a storage account named 'streamsdc123'. The left sidebar has a tree view with 'Storage account' expanded, showing 'Advanced Threat Protection ...', 'Static website', 'Properties', 'Locks', 'Automation script', 'Blob service' (which is expanded), and 'Custom domain'. Under 'Blob service', 'Blobs' is selected. The main pane shows a table of blob containers. The table has columns: NAME, LAST MODIFIED, PUBLIC ACCESS L..., and LEASE STATE. One row is present: 'learn-container' was last modified on 1/29/2019, 3:42:37 PM, is private, and its lease state is available.

| NAME | LAST MODIFIED | PUBLIC ACCESS L... | LEASE STATE |
|-----------------|-----------------------|--------------------|-------------|
| learn-container | 1/29/2019, 3:42:37 PM | Private | Available |

5. Select the **learn-container** container you created. It should be empty.
6. Select **Upload**. The **Upload blob** panel appears. Next to the **Files** text box, select the folder icon, and then select the JSON file.
7. Expand the **Advanced** options if they're not expanded already.
8. In the **Upload to folder** field, enter **input/[YYYY-MM-DD]**. Here, **YYYY-MM-DD** is the current date and needs to be entered using the data format you noted in the exercise "Configure the Azure Stream Analytics job input".
9. Leave the default values in the other fields.
10. Select **Upload**.

¹⁵ <https://portal.azure.com/>

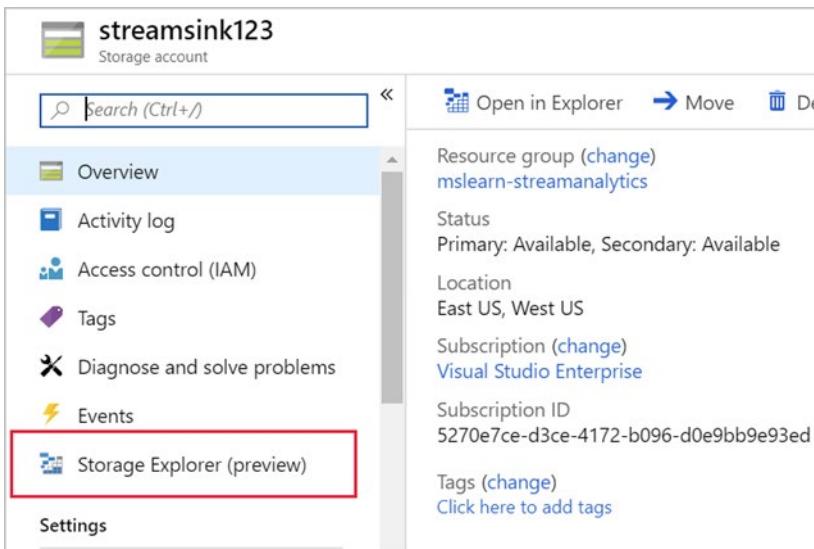


After the file is uploaded, you should see the **input** folder in the container. Select it to explore the blob hierarchy and see the data.

Set up the output Blob storage container

Next, set up the destination for the transformed data:

1. Go to your destination Blob storage account (**streamsink**).
2. From the choices on the overview page, select **Storage Explorer (preview)**.



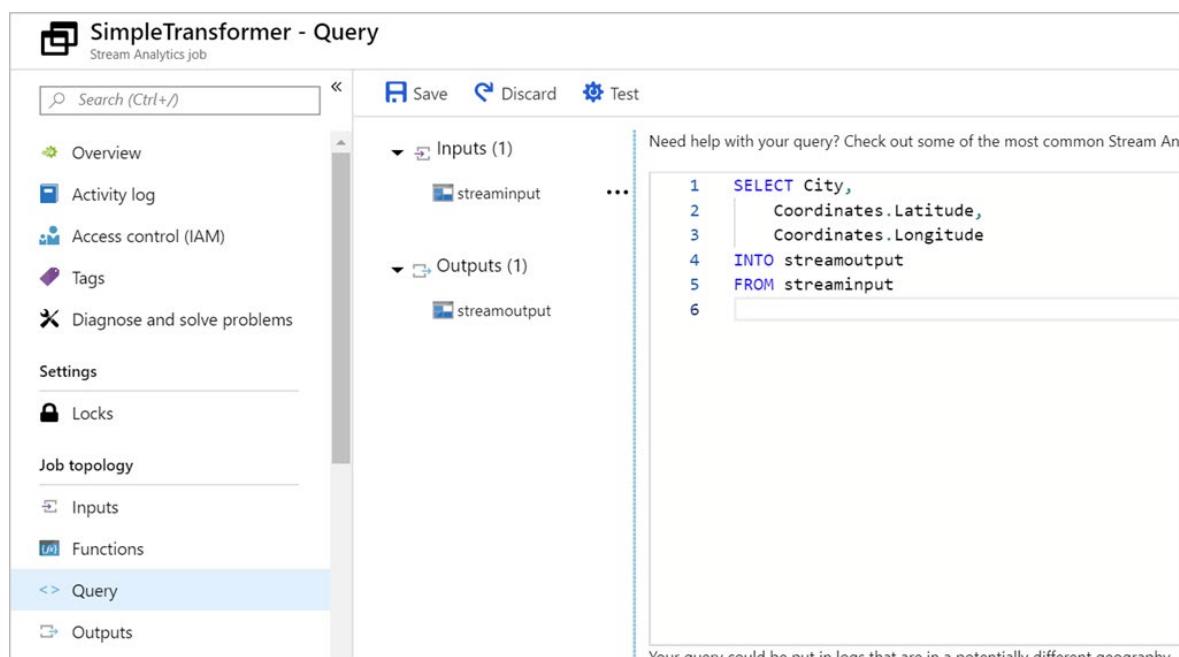
3. Select the container you created.
4. From the menu above the container details, select **New Folder**. If you don't see this option, open the **More** list to find it. The **Create New Virtual Directory** panel appears.
5. For the folder **Name**, enter **output**, and then select **OK**. Here, you're creating a placeholder. Azure won't show the folder until you add a file to it.

Write the transformation query

Now you're ready to write your transformation query. You'll need to pull the coordinates from the input data and write them to the output. You'll do that by using a `SELECT` statement. Find the query options online or by using the link in this module's summary.

1. Use the search field to find your Stream Analytics job in the Azure portal. Its name is **SimpleTransformer**.
2. Under **Job topology**, select **Query**.
3. In the **Query** pane, add this query:

```
SELECT City,  
       Coordinates.Latitude,  
       Coordinates.Longitude  
  INTO streamoutput  
 FROM streaminput
```



4. Select **Save**.

Test the query

Before you run a job, it's a good idea to test the query to make sure it does what you want. In the Azure portal, go to your Stream Analytics job. Under **Job topology**, select **Query > Test**. To start the test, just upload sample data for the query.

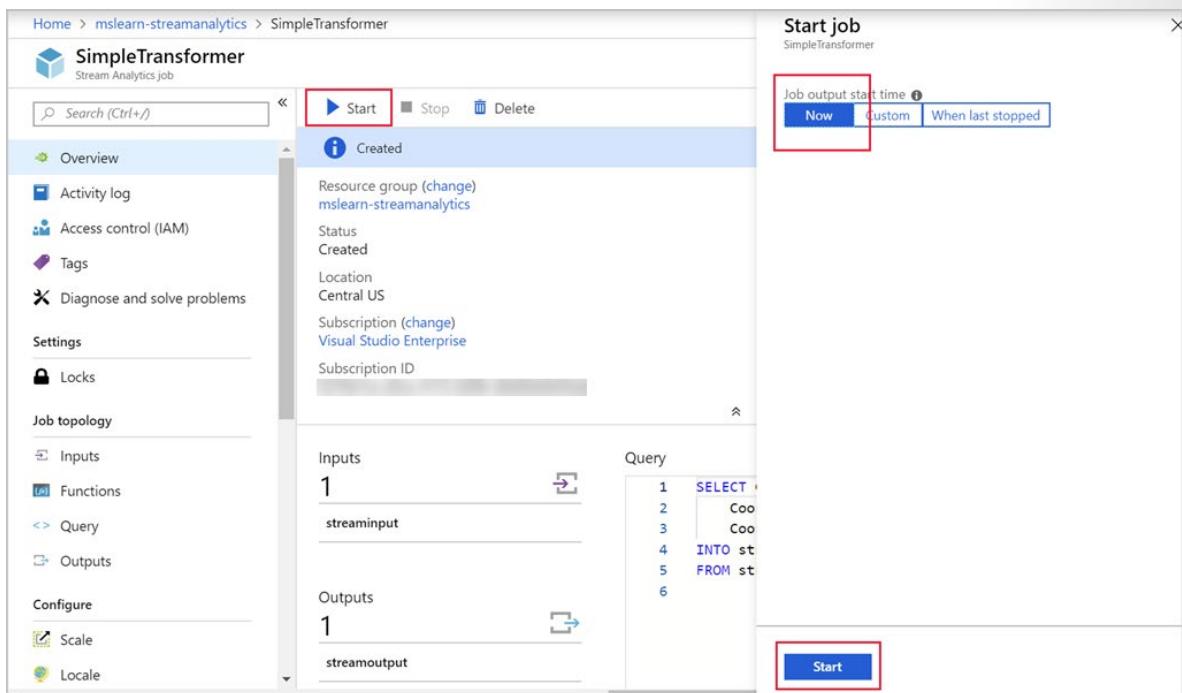
Exercise - Start an Azure Stream Analytics job

Now that you've created and set up a job, run it to produce an output.

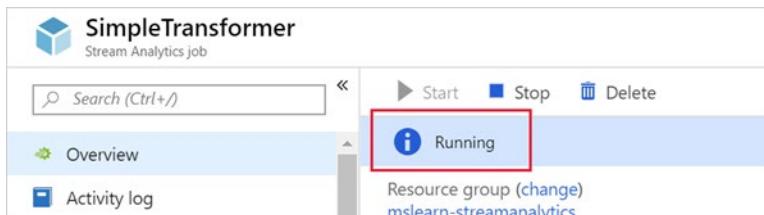
Perform these steps in the Azure portal.

1. On your job menu, select **Overview** to go back to the main overview page.

2. From the top menu bar, select **Start**.
3. Select **Now > Start**.



The job should transition to the **Starting** state. After a few seconds, it will move to the **Running** state. The job will run for a few minutes, and then finish in the **Completed** state. You can watch these transitions on the job's **Overview** page.

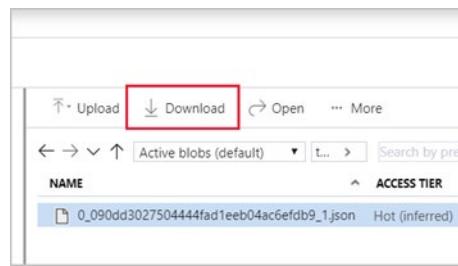


Exercise - View the results from an Azure Stream Analytics job

After a Stream Analytics job is completed, you can view the results in the Azure portal. On the job's **Overview** pane, you see the status information, the location, and resource group where the service is provisioned, and the subscription details. Here, you can also confirm when the service was created and started.

To see the job's results, perform the following steps.

1. In the Azure portal, go to your output storage account (**streamsink**).
2. Select **Storage Explorer (preview)**.
3. On the right, under **BLOB CONTAINERS**, open your container (**learn-container**).
4. Go to the **output** folder, and select **Download**.



When you open the file, you should see something like this.

```
```json
{
 "city" : "Reykjavik",
 "latitude" : 64,
 "longitude" : 21
}
```

```

Monitor and troubleshoot Azure Stream Analytics jobs

Monitoring is a key part of any mission-critical workload. It helps to proactively detect and prevent issues that might otherwise cause application or service downtime.

You can monitor Azure Stream Analytics jobs by using several tools:

- An activity log for each running job
- Real-time dashboards that show service and application health trends
- Alerts on issues in applications or services
- Diagnostic logs

Activity logs

The Stream Analytics activity log provides details about each job you run. This low-level troubleshooting tool can help you identify issues with data sources, outputs, or transformation queries.

Each job you create has an activity log. In the log, expand each job, and then select an event to see details in the JSON file.

The screenshot shows the 'tutorial - Activity log' page in the Azure Stream Analytics jobs section. The left sidebar includes options like 'Create a resource', 'Home', 'Dashboard', 'All services', 'FAVORITES' (with 'All resources' selected), 'Resource groups', 'App Services', 'Function Apps', 'SQL databases', 'Azure Cosmos DB', 'Virtual machines', 'Load balancers', and 'Storage accounts'. The main content area displays a table of activity logs with columns: OPERATION NAME, STATUS, TIME, TIME STAMP, and SUBSCRIPTION. The table shows seven entries all related to 'Start Stream Analytics Job' with various statuses (Succeeded, Started, Accepted) and times (within the last 6 hours). Filter options at the top include 'Subscription : Free Trial', 'Timespan : Last 6 hours', 'Event severity : All', 'Resource group : free', and 'Resource : tutorial'.

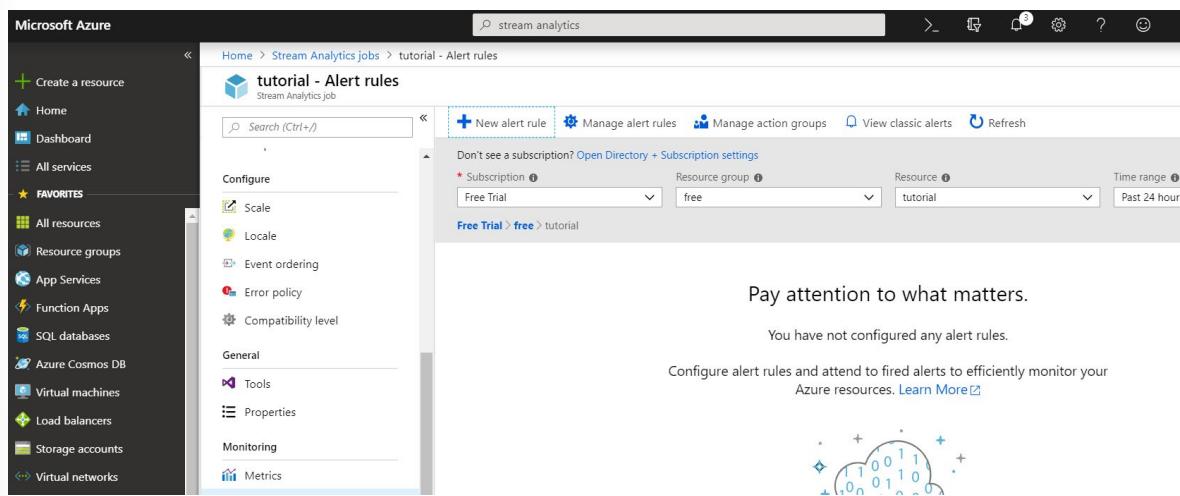
Dashboards

Dashboards show key health metrics for your Stream Analytics jobs. To view a live dashboard, go to the Azure portal, select your Stream Analytics job, and under **Monitoring**, select **Metrics**.

The screenshot shows the 'tutorial - Metrics' page in the Azure Stream Analytics jobs section. The left sidebar is identical to the previous screenshot. The main content area shows a 'Configure' section with options like 'Scale', 'Locale', 'Event ordering', 'Error policy', 'Compatibility level', 'General', 'Tools', 'Properties', and 'Monitoring'. Under 'Monitoring', the 'Metrics' tab is selected. A 'New chart' section allows adding a metric (selected as 'tutorial'), applying a filter, splitting the chart, and setting an alert rule. A dropdown menu for 'Select metric' lists various metrics: Backlogged Input Events, Data Conversion Errors, Early Input Events, Failed Function Requests, Function Events, Function Requests, Input Deserialization Errors, and Input Event Rates. The Y-axis scale ranges from 50 to 100.

Alerts

To proactively detect issues, you can set up Stream Analytics to fire alerts based on various metrics and thresholds. To set up alerts in the Azure portal, in your Stream Analytics job, under **Monitoring**, select **Alert rules > New alert rule**.



Microsoft Azure

Home > Stream Analytics jobs > tutorial - Alert rules

tutorial - Alert rules

Stream Analytics job

Search (Ctrl+)

+ New alert rule Manage alert rules Manage action groups View classic alerts Refresh

Configure

- Scale
- Locale
- Event ordering
- Error policy
- Compatibility level

General

- Tools
- Properties

Monitoring

- Metrics

Don't see a subscription? Open Directory + Subscription settings

Subscription: Free Trial Resource group: free Resource: tutorial Time range: Past 24 hours

Free Trial > free > tutorial

Pay attention to what matters.

You have not configured any alert rules.

Configure alert rules and attend to fired alerts to efficiently monitor your Azure resources. [Learn More](#)

Free Trial > free > tutorial

As you're setting up your rules, you can choose to send alerts by email, SMS, or voicemail. You can also use alerts to trigger workflows.

here' and a table for ACTION GROUP NAME and ACTION GROUP TYPE." data-bbox="157 386 880 665"/>

Home > Stream Analytics jobs > tutorial - Alert rules > Create rule

Create rule

Rules management

* RESOURCE

tutorial

HIERARCHY

Free Trial > free

Select

* CONDITION

No condition defined, click on 'Add condition' to select a signal and define its logic

Add condition

ACTION GROUPS

Notify your team via email and text messages or automate actions using webhooks, runbooks, functions, logic apps or integrating with external ITSM solutions. Learn more [here](#)

| ACTION GROUP NAME | ACTION GROUP TYPE |
|-------------------|-------------------|
| | |

Diagnostic logs

Diagnostic logging is a key part of operational infrastructure. Use diagnostic logs to help find root-cause issues in production deployments. You can conveniently deliver diagnostic logs to various sinks or destinations for root-cause analysis.

Stream Analytics diagnostics is turned off by default. In the Azure portal, you can turn it on when you need it. In the Stream Analytics job, under **Monitoring**, select **Diagnostic logs**.

You can persist diagnostics settings in an Azure Storage account or send them to Azure Event Hubs or Azure Log Analytics. Generate diagnostics logs for job execution or job authoring.

The screenshot shows the 'Diagnostics settings' page for a Stream Analytics job named 'tutorial - Diagnostics logs'. The left sidebar lists various Azure services, and the top navigation bar shows the full URL: Home > Stream Analytics jobs > tutorial - Diagnostics logs > Diagnostics settings. The main area contains fields for 'Name' (with a placeholder 'tutorial'), and checkboxes for 'Archive to a storage account', 'Stream to an event hub', and 'Send to Log Analytics'. Below these are sections for 'LOG' (Execution, Authoring) and 'METRIC' (AllMetrics), each with its own checkbox.

Knowledge check

Question 1

Which job input consumes data streams from applications at low latencies and high throughput?

- Azure Blob storage
- Azure Event Hubs
- Azure IoT Hub

Question 2

The Stream Analytics query language is a subset of which query language?

- T-SQL
- WQL
- JSON

Summary

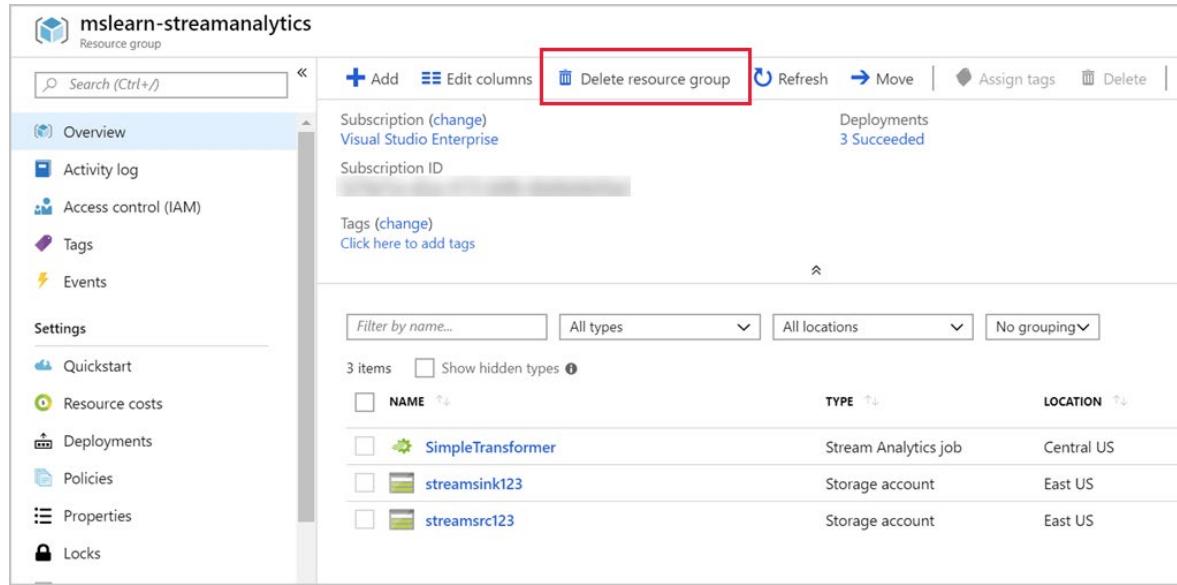
In this module, you learned how to create an Azure Stream Analytics job, set up an input, write a transformation query, and set up an output. You used Azure Storage as your source and destination, and you

created a transformation query to produce some basic results. You also learned how to start a Stream Analytics job and view the job results.

Clean up

To avoid unnecessary charges, you'll want to delete all the resources you created for this module. That includes the two storage accounts and the Stream Analytics job. You can delete these accounts individually, but it's easier to delete the resource group **mslearn-streamanalytics**:

1. Use the search field to find the resource group.
2. Select the group, and then select **Delete resource group**.



The screenshot shows the Azure portal interface for the 'mslearn-streamanalytics' resource group. On the left, there's a navigation menu with options like Overview, Activity log, Access control (IAM), Tags, Events, Settings, Quickstart, Resource costs, Deployments, Policies, Properties, and Locks. The 'Overview' tab is selected. At the top right, there are buttons for Add, Edit columns, Delete resource group (which is highlighted with a red box), Refresh, Move, Assign tags, and Delete. Below these buttons, there's information about the subscription (Visual Studio Enterprise) and deployment status (3 succeeded). The main area displays a list of resources with columns for Name, Type, and Location. The listed resources are:

| NAME | TYPE | LOCATION |
|-------------------|----------------------|------------|
| SimpleTransformer | Stream Analytics job | Central US |
| streamsink123 | Storage account | East US |
| streamsrc123 | Storage account | East US |

Because you're deleting a whole set of resources together, Azure prompts you to confirm that you want to do this.

Learn more

To learn more about creating Stream Analytics jobs and queries, use these resources:

- **Supported output sinks for Stream Analytics¹⁶**
- **Stream Analytics query language¹⁷**

¹⁶ <https://docs.microsoft.com/azure/stream-analytics/stream-analytics-define-outputs>

¹⁷ <https://docs.microsoft.com/stream-analytics-query/stream-analytics-query-language-reference>

Module Lab information

Lab 10 - Real-time Stream Processing with Stream Analytics

- **Estimated Time:** 50 - 60 minutes
- **Lab files:** The files for this lab are located in the *Instructions\Labs\14* folder.

Lab overview

In this lab, students will learn how to process streaming data with Azure Stream Analytics. The student will ingest vehicle telemetry data into Event Hubs, then process that data in real time, using various windowing functions in Azure Stream Analytics. They will output the data to Azure Synapse Analytics. Finally, the student will learn how to scale the Stream Analytics job to increase throughput.

Lab objectives

After completing this lab, you will be able to:

- Use Stream Analytics to process real-time data from Event Hubs
- Use Stream Analytics windowing functions to build aggregates and output to Azure Synapse Analytics
- Scale the Azure Stream Analytics job to increase throughput through partitioning
- Repartition the stream input to optimize parallelization

Module summary

module-summary

In this module, students have learned the concepts of event processing and streaming data and how this applies to Azure Stream Analytics and Azure Event Hubs. An understanding has been created about how to set up a stream analytics job to stream data, and have learned how to manage and monitor a running job. The studens have been enabled how to facilitate reliable messaging for big data applications using Azure Event Hubs.

Learning objectives

In this module, you have learned to:

- Work with data streams by using Azure Stream Analytics
- Facilitate reliable messaging for Big Data applications using Azure Event Hubs
- Ingest data streams with Azure Stream Analytics

Post Course Review

After the course, consider visiting the website that explores a **Azure Stream Analytics solution patterns¹⁸** Azure Stream Analytics solution pattern and the associated documentation that goes into more depth about this architecture.

Important

Remember

Should you be working in your own subscription while running through this content, it is best practice at the end of a project to identify whether or not you still need the resources you created. Resources left running can cost you money.

You can delete resources one by one, or just delete the resource group to get rid of the entire set.

¹⁸ <https://docs.microsoft.com/en-us/azure/stream-analytics/stream-analytics-solution-patterns>

Answers

Question 2

By default, how many partitions will a new Event Hub have?

- 2
- 3
- 4

Explanation

Event Hubs default to 4 partitions. Partitions are the buckets within an Event Hub. Each publication will go into only one partition. Each consumer group may read from one or more than one partition.

Question 3

What is the maximum size for a single publication (individual or batch) that is allowed by Azure Event Hub?

- 256 KB
- 1 MB
- 2 MB

Explanation

The maximum size for a single publication (individual or batch) that is allowed by Azure Event Hub is 1 MB.

Question 1

Which of the following technologies typically provide an ingestion point for data streaming in an event processing solution that uses *static* data as a source?

- Azure IoT Hub
- Azure Blob storage
- Azure Event Hub

Explanation

That's the correct answer. Azure Blob storage provides an ingestion point for data streaming in an event processing solution that uses static data as a source.

Question 2

To consume processed event streaming data in near-real-time to produce dashboards containing rich visualizations, which of the following services should you use?

- Azure Cosmos DB
- Event Hubs
- Power BI

Explanation

Correct. Power BI provides a platform for visualizing and analyzing aggregated data in near-real-time. Azure Stream Analytics can target Power BI as an output destination. Processed data is passed into Power BI to facilitate near-real-time dashboard updates.

Question 1

Which job input consumes data streams from applications at low latencies and high throughput?

- Azure Blob storage
- Azure Event Hubs
- Azure IoT Hub

Explanation

Correct. Event Hubs consumes data streams from applications at low latencies and high throughput.

Question 2

The Stream Analytics query language is a subset of which query language?

- T-SQL
- WQL
- JSON

Explanation

Correct. The query language you use in Stream Analytics is based heavily on T-SQL.

Module 11 Create a Stream Processing Solution with Event Hubs and Azure Databricks

Module introduction

module-introduction

This module teaches the student how Structured Streaming helps you process streaming data in real time using Azure Databricks and Azure Event Hubs. This module enables you to aggregate data over windows of time and read and write streams to Azure Event Hubs.

Learning objectives

In this module, you will:

- Learn the key features and uses of Structured Streaming.
- Stream data from a file and write it out to a distributed file system.
- Use sliding windows to aggregate over chunks of data rather than all data.
- Apply watermarking to throw away stale old data that you do not have space to keep.
- Connect to Event Hubs read and write streams.

Process streaming data with Azure Databricks structured streaming

Introduction

Suppose you work in the analytics department of a large airline company. You're part of a team that's analyzing the reasons for flight delays based on real-time data from flights around the world. Your job is to analyze the continuous flow of incoming data and stream it to Azure Data Lake storage. That data includes details such as airline, reason for delays, and departure time. You're using Azure Event Hubs to process your data.

Learning objectives

In this module, you will:

- Learn the key features and uses of Structured Streaming.
- Stream data from a file and write it out to a distributed file system.
- Use sliding windows to aggregate over chunks of data rather than all data.
- Apply watermarking to throw away stale old data that you do not have space to keep.
- Connect to Event Hubs read and write streams.

Prerequisites

None

Describe Azure Databricks structured streaming

Apache Spark Structured Streaming is a fast, scalable, and fault-tolerant stream processing API. You can use it to perform analytics on your streaming data in near real time.

With Structured Streaming, you can use SQL queries to process streaming data in the same way that you would process static data. The API continuously increments and updates the final data.

Event Hubs and Spark Structured Streaming

Azure Event Hubs is a scalable real-time data ingestion service that processes millions of data in a matter of seconds. It can receive large amounts of data from multiple sources and stream the prepared data to Azure Data Lake or Azure Blob storage.

Azure Event Hubs can be integrated with Spark Structured Streaming to perform processing of messages in near real time. You can query and analyze the processed data as it comes by using a Structured Streaming query and Spark SQL.

Streaming concepts

Stream processing is where you continuously incorporate new data into Data Lake storage and compute results. The streaming data comes in faster than it can be consumed when using traditional batch-related processing techniques. A stream of data is treated as a table to which data is continuously appended.

Examples of such data include bank card transactions, Internet of Things (IoT) device data, and video game play events.

A streaming system consists of:

- Input sources such as Kafka, Azure Event Hubs, IoT Hub, files on a distributed system, or TCP-IP sockets
- Stream processing using Structured Streaming, forEach sinks, memory sinks, etc.

Perform stream processing using structured streaming

In this unit, you need to complete the exercises within a Databricks Notebook. To begin, you need to have access to an Azure Databricks workspace. You also need an Azure Event Hubs instance in your Azure subscription.

If you are using a pre-provisioned environment, you can skip to the bottom of the page to Clone the Databricks archive.

Unit Pre-requisites

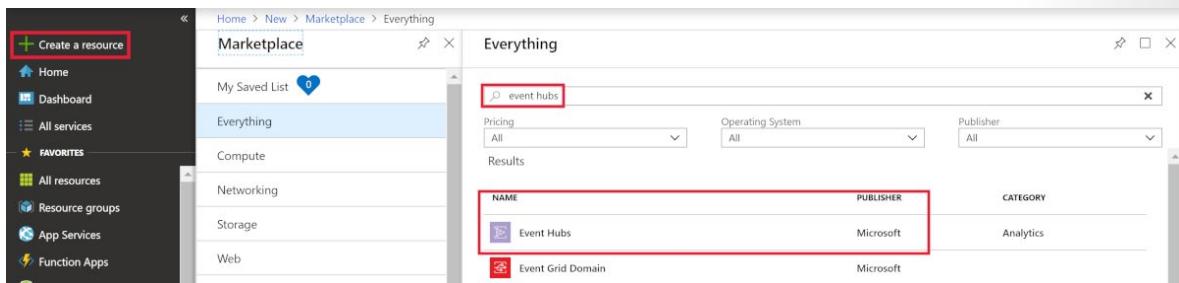
Microsoft Azure Account: You will need a valid and active Azure account for the Azure labs. If you do not have one, you can sign up for a [free trial](#)¹

- If you are a Visual Studio Active Subscriber, you are entitled to Azure credits per month. You can refer to this [link](#)² to find out more including how to activate and start using your monthly Azure credit.
- If you are not a Visual Studio Subscriber, you can sign up for the FREE [Visual Studio Dev Essentials](#)³ program to create Azure free account.

The first step toward using Spark Structured Streaming is setting up Azure Event Hubs in your Azure subscription.

Create an Event Hubs namespace

1. In the Azure portal, select **+ Create a resource**. Enter **event hubs** into the **Search the Marketplace** box, select **Event Hubs** from the results, and then select **Create**.



2. In the **Create Namespace** pane, enter the following information:

- **Subscription:** Select the subscription group you're using for this module.

¹ <https://azure.microsoft.com/free/>

² <https://azure.microsoft.com/pricing/member-offers/msdn-benefits-details/>

³ <https://www.visualstudio.com/dev-essentials/>

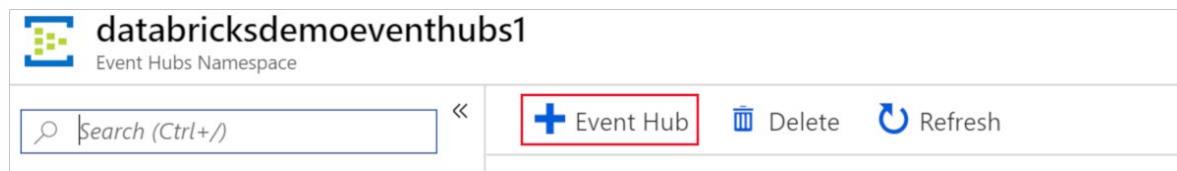
- **Resource group:** Choose your module resource group.
- **Namespace name:** Enter a unique name, such as **databricksdemoeventhubs**. Uniqueness will be indicated by a green check mark.
- **Location:** Select the location you're using for this module.
- **Pricing tier:** Select **Basic**.

Select **Review + create**, then select **Create**.

The screenshot shows the 'Create Namespace' wizard in the Azure portal. The 'Basics' tab is selected. In the 'PROJECT DETAILS' section, the 'Subscription' dropdown is set to a placeholder value. The 'Resource group' dropdown also has a placeholder value, with a 'Create new' link below it. In the 'INSTANCE DETAILS' section, the 'Namespace name' field contains 'databricksdemoeventhubs3' followed by '.servicebus.windows.net', with a green checkmark indicating uniqueness. The 'Location' is set to 'West US'. The 'Pricing tier' is set to 'Basic (1 Consumer group, 100 Brokered connections)'. The 'Throughput Units' slider is set to 1. At the bottom, there are buttons for 'Review + create', '< Previous', and 'Next: Features >'.

Create an event hub

1. After your Event Hubs namespace is provisioned, browse to it and add a new event hub by selecting the **+ Event Hub** button on the toolbar.



2. On the **Create Event Hub** pane, enter:

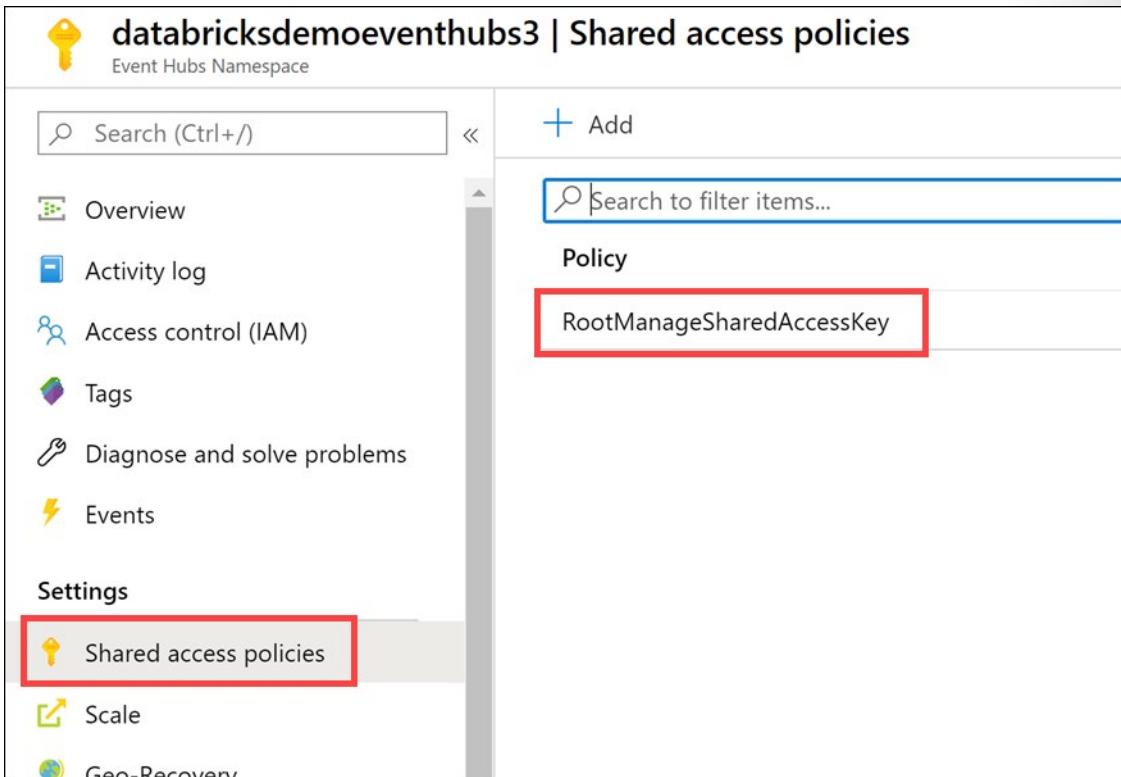
- **Name:** Enter **databricks-demo-eventhub**.

- **Partition Count:** Enter 2.

Select **Create**.

Copy the connection string primary key for the shared access policy

1. On the left-hand menu in your Event Hubs namespace, select **Shared access policies** under **Settings**, then select the **RootManageSharedAccessKey** policy.



The screenshot shows the Azure portal interface for managing an Event Hubs namespace named "databricksdemoeventhubs3". The left sidebar has a "Settings" section with three options: "Shared access policies" (highlighted with a red box), "Scale", and "Geo-Recovery". The main content area is titled "Shared access policies" and shows a list of policies. One policy, "RootManageSharedAccessKey", is highlighted with a red box. A search bar at the top right says "Search to filter items...".

2. Copy the connection string for the primary key by selecting the copy button.

SAS Policy: RootManageSharedAccessKey

Save Discard Delete ...

Manage

Send

Listen

Primary key

GuAj2zdcehfXNJXjeeXB6eEOWR4xaNcGwra0LaG9N... [Copy](#)

Secondary key

3V5I+g4CMFkw5SPhmHh9uOFWIP2nEe7RfomOJOX... [Copy](#)

Connection string-primary key

Endpoint=sb://databricksdemoeventhubs3.serviceb... [Copy](#)

Connection string-secondary key

Endpoint=sb://databricksdemoeventhubs3.serviceb... [Copy](#)

3. Save the copied primary key to Notepad.exe or another text editor for later reference.

Deploy an Azure Databricks workspace

1. Click the following button to open the Azure Resources Manager template in the Azure portal.
Deploy Databricks from the Azure Resource Manager Template⁴
2. Provide the required values to create your Azure Databricks workspace:
 - **Subscription:** Choose the Azure Subscription in which to deploy the workspace.
 - **Resource Group:** Leave at Create new and provide a name for the new resource group.
 - **Location:** Select a location near you for deployment. For the list of regions supported by Azure Databricks, see **Azure services available by region⁵**.
 - **Workspace Name:** Provide a name for your workspace.
 - **Pricing Tier:** Ensure premium is selected.
3. Accept the terms and conditions.
4. Select Purchase.
5. The workspace creation takes a few minutes. During workspace creation, the portal displays the Submitting deployment for Azure Databricks tile on the right side. You may need to scroll right on

⁴ <https://portal.azure.com/#create/Microsoft.Template/uri/https%3A%2F%2Fraw.githubusercontent.com%2FAzure%2Fazure-quickstart-templates%2Fmaster%2F101-databricks-workspace%2Fazuredeploy.json>

⁵ <https://azure.microsoft.com/regions/services/>

your dashboard to see the tile. There is also a progress bar displayed near the top of the screen. You can watch either area for progress.

Create a cluster

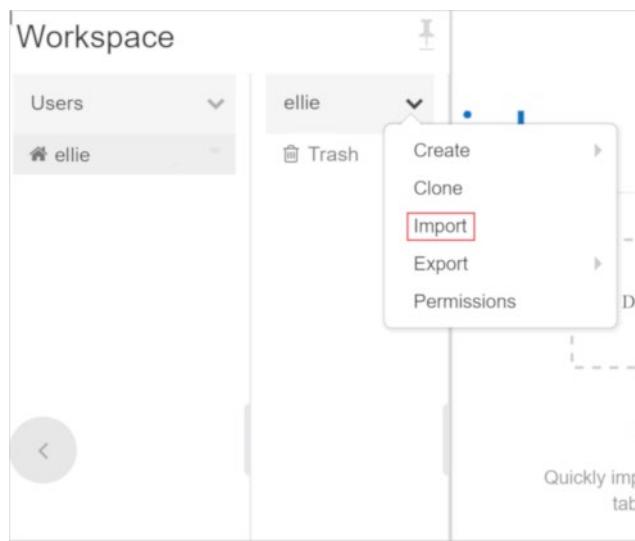
1. When your Azure Databricks workspace creation is complete, select the link to go to the resource.
2. Select **Launch Workspace** to open your Databricks workspace in a new tab.
3. In the left-hand menu of your Databricks workspace, select **Clusters**.
4. Select **Create Cluster** to add a new cluster.

The screenshot shows the 'Create Cluster' dialog box. At the top, it says 'Create Cluster' and 'New Cluster'. Below that, there are fields for 'Cluster Name' (set to 'Test Cluster'), 'Cluster Mode' (set to 'Standard'), 'Pool' (set to 'None'), and 'Databricks Runtime Version' (set to 'Runtime: 8.0 (Scala 2.12, Spark 3.1.1)'). A note below says 'Databricks Runtime 8.x uses Delta Lake as the default table format.' Under 'Autopilot Options', two checkboxes are checked: 'Enable autoscaling' and 'Terminate after 120 minutes of inactivity'. Below that, under 'Worker Type', 'Standard_DS3_v2' is selected with '14.0 GB Memory, 4 Cores, 0.75 DBU'. To the right, 'Min Workers' is set to '2' and 'Max Workers' is set to '8'. A checkbox for 'Spot instances' is present. Under 'Driver Type', 'Same as worker' is selected with '14.0 GB Memory, 4 Cores, 0.75 DBU'.

5. Enter a name for your cluster. Use your name or initials to easily differentiate your cluster from your coworkers.
6. Select the **Databricks RuntimeVersion**. We recommend the latest runtime and **Scala 2.12**.
7. Select the default values for the cluster configuration.
8. Select **Create Cluster**.

Clone the Databricks archive

1. If you do not currently have your Azure Databricks workspace open: in the Azure portal, navigate to your deployed Azure Databricks workspace and select **Launch Workspace**.
2. In the left pane, select **Workspace > Users**, and select your username (the entry with the house icon).
3. In the pane that appears, select the arrow next to your name, and select **Import**.



4. In the **Import Notebooks** dialog box, select the URL and paste in the following URL:

<https://github.com/solliancenet/microsoft-learning-paths-databricks-notebooks/blob/master/data-engineering/DBC/10-Structured-Streamingdbc?raw=true>

5. Select **Import**.
6. Select the **10-Structured-Streaming** folder that appears.

Complete the following notebook

Open the **1.Structured-Streaming-Concepts** notebook. Make sure you attach your cluster to the notebook before following the instructions and running the cells within.

Within the notebook, you will:

- Stream data from a file and write it out to a distributed file system
- List active streams
- Stop active streams

Please note: Execute one cell at a time. Do *not* select Run All at the top of the notebook.

Reading a stream

The method `SparkSession.readStream` returns a `DataStreamReader` used to configure the stream.

There are a number of key points to the configuration of a `DataStreamReader`:

- The schema

- The type of stream: Files, Kafka, TCP/IP, etc
- Configuration specific to the type of stream
 - For files, the file type, the path to the files, max files, etc.
 - For TCP/IP the server's address, port number, etc.
 - For Kafka the server's address, port, topics, partitions, etc.

The Schema

Every streaming DataFrame must have a schema - the definition of column names and data types. Some sources such as Pub/Sub sources like Kafka and Event Hubs define the schema for you. For file-based streaming sources, the schema must be user-defined.

Why must a schema be specified for a streaming DataFrame?

To say that another way...

Why are streaming DataFrames unable to infer/read a schema?

If you have enough data, you can infer the schema. If you don't have enough data you run the risk of miss-inferring the schema.

For example, you think you have all integers but the last value contains "1.123" (a float) or "snoopy" (a string). With a stream, we have to assume we don't have enough data because we are starting with zero records.

And unlike reading from a table or parquet file, there is nowhere from which to "read" the stream's schema.

For this reason, we must specify the schema manually:

```
# Here we define the schema using a DDL-formatted string (the SQL Data Definition Language).
dataSchema = "Recorded_At timestamp, Device string, Index long, Model string, User string, _corrupt_record String, gt string, x double, y double, z double"
```

Configuring a File Stream

In our example below, we will be consuming files written continuously to a pre-defined directory. To control how much data is pulled into Spark at once, we can specify the option `maxFilesPerTrigger`.

In our example below, we will be reading in only one file for every trigger interval:

```
.option("maxFilesPerTrigger", 1)
```

Both the location and file type are specified with the following call, which itself returns a `DataFrame`:

```
.json(dataPath)
```

```
dataPath = "dbfs:/mnt/training/definitive-guide/data/activity-data-stream.json"
initialDF = (spark
    .readStream
    .option("maxFilesPerTrigger", 1)           # Returns DataStreamReader
    per trigger                                # Force processing of only 1 file
    .schema(dataSchema)                        # Required for all streaming
    DataFrames
    .json(dataPath)                            # The stream's source directory
and file type
)
```

And with the initial DataFrame, we can apply some transformations:

```
streamingDF = (initialDF
    .withColumnRenamed("Index", "User_ID")   # Pick a "better" column name
    .drop("_corrupt_record")                 # Remove an unnecessary column
)
```

Streaming DataFrames

Other than the call to `spark.readStream`, it looks just like any other DataFrame. But is it a "streaming" DataFrame?

You can differentiate between a "static" and "streaming" DataFrame with the following call:

```
# Static vs Streaming?
streamingDF.isStreaming
```

Unsupported Operations

Most operations on a "streaming" DataFrame are identical to a "static" DataFrame. There are some exceptions to this. One such example would be to sort our never-ending stream by `Recorded_At`:

```
from pyspark.sql.functions import col

try:
    sortedDF = streamingDF.orderBy(col("Recorded_At").desc())
    display(sortedDF)
except:
    print("Sorting is not supported on an unaggregated stream")
```

Sorting is one of a handful of operations that is either too complex or logically not possible to do with a stream.

For more information on this topic, see the **Structured Streaming Programming Guide / Unsupported Operations**⁶.

We will see in the following unit how we can sort an **aggregated** stream.

⁶ <https://spark.apache.org/docs/latest/structured-streaming-programming-guide.html#unsupported-operations>

Writing a stream

The method `DataFrame.writeStream` returns a `DataStreamWriter` used to configure the output of the stream.

There are a number of parameters to the `DataStreamWriter` configuration:

- Query's name (optional) - This name must be unique among all the currently active queries in the associated `SQLContext`.
- Trigger (optional) - Default value is `ProcessingTime(0)` and it will run the query as fast as possible.
- Checkpointing directory (optional for pub/sub sinks)
- Output mode
- Output sink
- Configuration specific to the output sink, such as:
 - The host, port and topic of the receiving Kafka server
 - The file format and final destination of files
 - A [custom sink via `writeStream.foreach\(...\)`](https://spark.apache.org/docs/latest/api/python/pyspark.sql.html?highlight=foreach#pyspark.sql.streaming.DataStreamWriter.foreach)

Once the configuration is completed, we can trigger the job with a call to `.start()`

Triggers

The trigger specifies when the system should process the next set of data.

| Trigger Type | Example | Notes |
|--|--|--|
| Unspecified | | <i>DEFAULT</i> - The query will be executed as soon as the system has completed processing the previous query |
| Fixed interval micro-batches | <code>.trigger(Trigger.ProcessingTime("6 hours"))</code> | The query will be executed in micro-batches and kicked off at the user-specified intervals |
| One-time micro-batch | <code>.trigger(Trigger.Once())</code> | The query will execute <i>only one</i> micro-batch to process all the available data and then stop on its own |
| Continuous w/fixed checkpoint interval | <code>.trigger(Trigger.Continuous("1 second"))</code> | The query will be executed in a low-latency, continuous processing mode (http://spark.apache.org/docs/latest/structured-streaming-programming-guide.html#continuous-processing). <i>EXPERIMENTAL</i> in 2.3.2 |

In the example below, you will be using a fixed interval of 3 seconds:

```
.trigger(Trigger.ProcessingTime("3 seconds"))
```

Checkpointing

A **checkpoint** stores the current state of your streaming job to a reliable storage system such as Azure Blob Storage or HDFS. It does not store the state of your streaming job to the local file system of any node in your cluster.

Together with write ahead logs, a terminated stream can be restarted and it will continue from where it left off.

To enable this feature, you only need to specify the location of a checkpoint directory:

```
.option("checkpointLocation", checkpointPath)
```

Points to consider:

- If you do not have a checkpoint directory, when the streaming job stops, you lose all state around your streaming job and upon restart, you start from scratch.
- For some sinks, you will get an error if you do not specify a checkpoint directory:
analysisException: 'checkpointLocation must be specified either through option("checkpointLocation", ...)...
- Also note that every streaming job should have its own checkpoint directory: no sharing.

Output modes

| Mode | Example | Notes |
|-----------------|---------------------------------------|--|
| Complete | <code>.outputMode ("complete")</code> | The entire updated Result Table is written to the sink. The individual sink implementation decides how to handle writing the entire table. |
| Append | <code>.outputMode ("append")</code> | Only the new rows appended to the Result Table since the last trigger are written to the sink. |
| Update | <code>.outputMode ("update")</code> | Only the rows in the Result Table that were updated since the last trigger will be outputted to the sink. Since Spark 2.1.1 |

In the example below, we are writing to a Parquet directory which only supports the `append` mode:

```
dsw.outputMode ("append")
```

Output sinks

`DataStreamWriter.format` accepts the following values, among others:

| Output Sink | Example | Notes |
|-------------|---|---|
| File | <code>dsw.format ("parquet"),
dsw.format ("csv") ...</code> | Dumps the Result Table to a file. Supports Parquet, json, csv, etc. |

| Output Sink | Example | Notes |
|----------------|------------------------------------|---|
| Kafka | dsw.format("kafka") | Writes the output to one or more topics in Kafka |
| Console | dsw.format("console") | Prints data to the console (useful for debugging) |
| Memory | dsw.format("memory") | Updates an in-memory table, which can be queried through Spark SQL or the DataFrame API |
| foreach | dsw.foreach(writer: ForeachWriter) | This is your "escape hatch", allowing you to write your own type of sink. |
| Delta | dsw.format("delta") | A proprietary sink |

In the example below, we will be appending files to a Parquet directory and specifying its location with this call:

```
.format("parquet").start(outputPathDir)
```

Streaming example

In the notebook, we write data from a streaming query to `outputPathDir`.

There are a couple of things to note in the example below:

1. We are giving the query a name via the call to `.queryName`
2. Spark begins running jobs once we call `.start`
3. The call to `.start` returns a `StreamingQuery` object

```
basePath = userhome + "/structured-streaming-concepts/python" # A working
directory for our streaming app
dbutils.fs.mkdirs(basePath) # Make sure
that our working directory exists
outputPathDir = basePath + "/output.parquet" # A subdirec-
tory for our output
checkpointPath = basePath + "/checkpoint" # A subdirec-
tory for our checkpoint & W-A logs

streamingQuery = (streamingDF
our "streaming" DataFrame
    .writeStream
DataStreamWriter
    .queryName("stream_1p") # Name the
query
    .trigger(processingTime="3 seconds") # Configure
for a 3-second micro-batch
    .format("parquet") # Specify the
sink type, a Parquet file
    .option("checkpointLocation", checkpointPath) # Specify the
location of checkpoint files & W-A logs
    .outputMode("append") # Write only
new data to the "file"
    .start(outputPathDir) # Start the
```

```
    job, writing to the specified directory
)
```

Managing streaming queries

When a query is started, the `StreamingQuery` object can be used to monitor and manage the query.

| Method | Description |
|---------------------------------|---|
| <code>id</code> | get unique identifier of the running query that persists across restarts from checkpoint data |
| <code>runId</code> | get unique id of this run of the query, which will be generated at every start/restart |
| <code>name</code> | get name of the auto-generated or user-specified name |
| <code>explain()</code> | print detailed explanations of the query |
| <code>stop()</code> | stop query |
| <code>awaitTermination()</code> | block until query is terminated, with <code>stop()</code> or with error |
| <code>exception</code> | exception if query terminated with error |
| <code>recentProgress</code> | array of most recent progress updates for this query |
| <code>lastProgress</code> | most recent progress update of this streaming query |

The display function

Within the Databricks notebooks, we can use the `display()` function to render a live plot

When you pass a "streaming" `DataFrame` to `display()`:

- A "memory" sink is being used
- The output mode is complete
- The query name is specified with the `streamName` parameter
- The trigger is specified with the `trigger` parameter
- The checkpointing location is specified with the `checkpointLocation`

```
display(myDF, streamName = "myQuery")
```

We just programmatically stopped our only streaming query in the previous cell. In the cell below, `display` will automatically start our streaming `DataFrame`, `streamingDF`. We are passing `stream_2p` as the name for this newly started stream.

End-to-end Fault Tolerance

Structured Streaming ensures end-to-end exactly-once fault-tolerance guarantees through *checkpointing* and **Write Ahead Logs**⁷.

⁷ https://en.wikipedia.org/wiki/Write-ahead_logging

Structured Streaming sources, sinks, and the underlying execution engine work together to track the progress of stream processing. If a failure occurs, the streaming engine attempts to restart and/or reprocess the data.

For best practices on recovering from a failed streaming query see docs.

This approach *only* works if the streaming source is replayable. To ensure fault-tolerance, Structured Streaming assumes that every streaming source has offsets, akin to:

- **Kafka message offsets⁸**
- **Event Hubs offsets⁹**

At a high level, the underlying streaming mechanism relies on a couple approaches:

- First, Structured Streaming uses checkpointing and write-ahead logs to record the offset range of data being processed during each trigger interval.
- Next, the streaming sinks are designed to be *idempotent*—that is, multiple writes of the same data (as identified by the offset) do *not* result in duplicates being written to the sink.

Taken together, replayable data sources and idempotent sinks allow Structured Streaming to ensure **end-to-end, exactly-once semantics** under any failure condition.

Summary

We use `readStream` to read streaming input from a variety of input sources and create a DataFrame.

Nothing happens until we invoke `writeStream` or `display`.

Using `writeStream` we can write to a variety of output sinks. Using `display` we draw LIVE bar graphs, charts and other plot types in the notebook.

Continue to the next step

After you've completed the notebook, return to this screen, and continue to the next step.

Work with Time Windows

In your Azure Databricks workspace, open the **10-Structured-Streaming** folder that you imported within your user folder.

Open the **2.Time-Windows** notebook. Make sure you attach your cluster to the notebook before following the instructions and running the cells within.

Within the notebook, you will:

- Use sliding windows to aggregate over chunks of data rather than all data
- Apply watermarking to throw away stale old data that you do not have space to keep
- Plot live graphs using `display`

Please note: Execute one cell at a time. Do *not* select Run All at the top of the notebook.

Streaming aggregations

Continuous applications often require near real-time decisions on real-time, aggregated statistics.

⁸ https://kafka.apache.org/documentation/#intro_topics

⁹ <https://docs.microsoft.com/azure/event-hubs/event-hubs-features>

Some examples include:

- Aggregating errors in data from IoT devices by type
- Detecting anomalous behavior in a server's log file by aggregating by country.
- Doing behavior analysis on instant messages via hash tags.

However, in the case of streams, you generally don't want to run aggregations over the entire dataset.

What problems might you encounter if you aggregate over a stream's entire dataset?

While streams have a definitive start, there conceptually is no end to the flow of data. Because there is no "end" to a stream, the size of the dataset grows in perpetuity. This means that your cluster will eventually run out of resources.

Instead of aggregating over the entire dataset, you can aggregate over data grouped by windows of time (say, every 5 minutes or every hour).

This is referred to as windowing.

Windowing

If we were using a static DataFrame to produce an aggregate count, we could use `groupBy()` and `count()`. Instead, we accumulate counts within a sliding window, answering questions like "How many records are we getting every second?"

Sliding windows

The windows overlap and a single event may be aggregated into multiple windows.

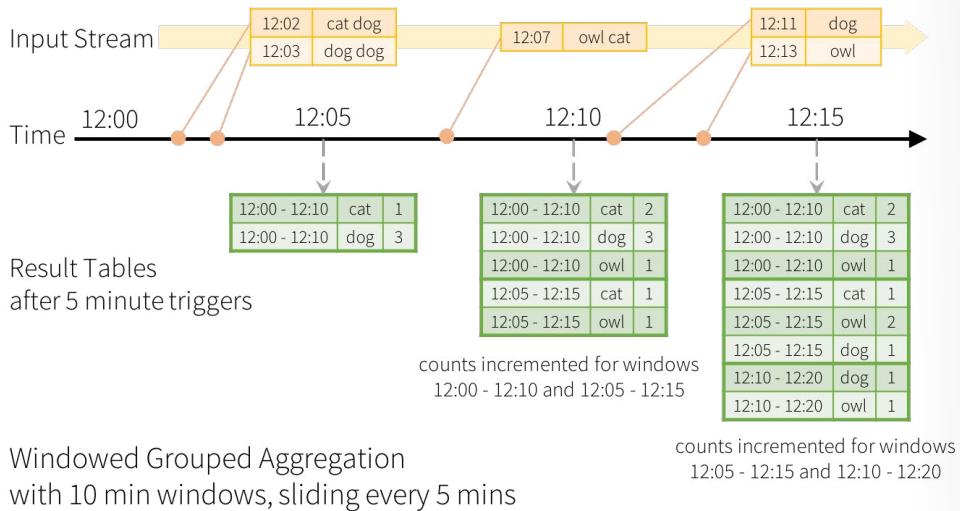
Tumbling Windows

The windows do not overlap and a single event will be aggregated into only one window.

The diagram below shows sliding windows.

The following illustration, from the **Structured Streaming Programming Guide**¹⁰ guide, helps us understanding how it works:

¹⁰ <https://spark.apache.org/docs/latest/structured-streaming-programming-guide.html>



Event Time vs Receive Time

Event Time is the time at which the event occurred in the real world.

Event Time is **NOT** something maintained by the Structured Streaming framework.

At best, Structured Streaming only knows about **Receipt Time** - the time a piece of data arrived in Spark.

What are some examples of Event Time? of Receipt Time?

Examples of Event Time

- The timestamp recorded in each record of a log file
- The instant at which an IoT device took a measurement
- The moment a REST API received a request

Examples of Receipt Time

- A timestamp added to a DataFrame the moment it was processed by Spark
- The timestamp extracted from an hourly log file's file name
- The time at which an IoT hub received a report of a device's measurement
- Presumably offset by some delay from when the measurement was taken

What are some of the inherent problems with using Receipt Time?

The main problem with using **Receipt Time** is going to be with accuracy. For example:

- The time between when an IoT device takes a measurement vs when it is reported can be off by several minutes.
 - This could have significant ramifications to security and health devices, for example
- The timestamp embedded in an hourly log file can be off by up to one hour making correlations to other events extremely difficult
- The timestamp added by Spark as part of a DataFrame transformation can be off by hours to weeks to months depending on when the event occurred and when the job ran

When might it be OK to use Receipt Time instead of Event Time?

When accuracy is not a significant concern - that is **Receipt Time** is close enough to **Event Time**

One example would be for IoT events that can be delayed by minutes but the resolution of your query is by days or months (close enough)

Windowed streaming example

```
from pyspark.sql.functions import window, col

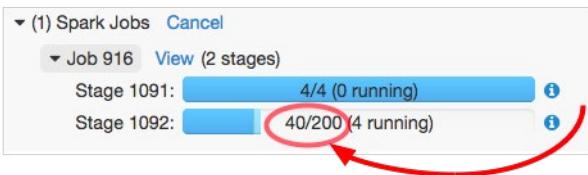
inputDF = (spark
    .readStream
    DataStreamReader
    .schema(jsonSchema)
    data
    .option("maxFilesPerTrigger", 1)
    a stream, one file at a time
    .json(inputPath)
    and returns a DataFrame
)

countsDF = (inputDF
    .groupBy(col("action"),
        window(col("time"), "1 hour"))
    .count()
    a count
    .select(col("window.start").alias("start"),
        col("action"),
        col("count"))
    .orderBy(col("start"), col("action"))
    action
)
```

Performance Considerations

If you run that query, as is, it will take a surprisingly long time to start generating data. What's the cause of the delay?

If you expand the **Spark Jobs** component, you'll see something like this:



It's our `groupBy().groupBy()` causes a *shuffle*, and, by default, Spark SQL shuffles to 200 partitions. In addition, we're doing a *stateful aggregation*: one that requires Structured Streaming to maintain and aggregate data over time.

When doing a stateful aggregation, Structured Streaming must maintain an in-memory *state map* for each window within each partition. For fault tolerance reasons, the state map has to be saved after a partition is processed, and it needs to be saved somewhere fault-tolerant. To meet those requirements, the Streaming API saves the maps to a distributed store. On some clusters, that will be HDFS. Azure Databricks uses the DBFS.

That means that every time it finishes processing a window, the Streaming API writes its internal map to disk. The write has some overhead, typically between 1 and 2 seconds.

What's the cause of the delay?

- `groupBy()` causes a **shuffle**
- By default, this produces **200 partitions**
- Plus a **stateful aggregation** to be maintained **over time**

This results in:

- Maintenance of an **in-memory state map** for **each window** within **each partition**
- Writing of the state map to a fault-tolerant store
 - On some clusters, that will be HDFS
 - Azure Databricks uses the DBFS
 - Around 1 to 2 seconds overhead

Shuffle Partition Best Practices

One way to reduce this overhead is to reduce the number of partitions Spark shuffles to.

In most cases, you want a 1-to-1 mapping of partitions to cores for streaming applications.

Run query with proper setting for shuffle partitions

Rerun the query below and notice the performance improvement.

Once the data is loaded, render a line graph with

- **Keys** is set to start
- **Series groupings** is set to action
- **Values** is set to count

```
spark.conf.set("spark.sql.shuffle.partitions", sc.defaultParallelism)  
display(countsDF)
```

Problem with generating many windows

We are generating a window for every 1 hour aggregate. *Every window* has to be separately persisted and maintained. Over time, this aggregated data will build up in the driver.

The end result being a massive slowdown if not an OOM Error.

How do we fix that problem?

One simple solution is to increase the size of our window (say, to 2 hours). That way, we're generating fewer windows.

But if the job runs for a long time, we're still building up an unbounded set of windows. Eventually, we could hit resource limits.

Watermarking

A better solution to the problem is to define a cut-off. A point after which Structured Streaming will commit windowed data to sink, or throw it away if the sink is console or memory as `display()` mimics.

That's what *watermarking* allows us to do.

Refining our previous example

Below is our previous example with watermarking.

We're telling Structured Streaming to keep no more than 2 hours of aggregated data.

```
watermarkedDF = (inputDF  
    .withWatermark("time", "2 hours")          # Specify a 2-hour watermark  
    .groupBy(col("action"),  
            window(col("time"), "1 hour"))      # ...then by a 1 hour window  
    .count()                                    # For each aggregate, produce  
    a count  
    .select(col("window.start").alias("start"), # Elevate field to column  
           col("action"),                      # Include count  
           col("count"))                      # Include action  
    .orderBy(col("start"), col("action"))       # Sort by the start time  
)  
display(watermarkedDF)                         # Start the stream and dis-  
play it
```

Example Details

In the example above,

- Data received 2 hour *past* the watermark will be dropped.
- Data received within 2 hours of the watermark will never be dropped.

More specifically, any data less than 2 hours behind the latest data processed till then is guaranteed to be aggregated.

However, the guarantee is strict only in one direction. Data delayed by more than 2 hours is not guaranteed to be dropped; it may or may not get aggregated.

The more delayed the data is, the less likely the engine is going to process it.

Continue to the next step

After you've completed the notebook, return to this screen, and continue to the next step.

Process data from Event Hubs with structured streaming

In your Azure Databricks workspace, open the **10-Structured-Streaming** folder that you imported within your user folder.

Open the **3.Streaming-With-Event-Hubs-Demo** notebook. Make sure you attach your cluster to the notebook before following the instructions and running the cells within.

Within the notebook, you will:

- Connect to Event Hubs and write a stream to your event hub
- Read a stream from your event hub
- Define a schema for the JSON payload and parse the data do display it within a table

Please note: Execute one cell at a time. Do *not* select Run All at the top of the notebook.

Azure Event Hubs

Azure Event Hubs is a fully managed, real-time data ingestion service. You can stream millions of events per second from any source to build dynamic data pipelines and immediately respond to business challenges.

It integrates seamlessly with a host of other Azure services.

Event Hubs can be used in a variety of applications such as:

- Anomaly detection (fraud/outliers)
- Application logging
- Analytics pipelines, such as clickstreams
- Archiving data
- Transaction processing
- User telemetry processing
- Device telemetry streaming

- **Live dashboarding**

Write a stream to Event Hubs to produce a stream

In the notebook, you execute the following cell (after you retrieve your Event Hubs connection string, per the instructions in the notebook):

```
%python

# For 2.3.15 version and above, the configuration dictionary requires that
connection string be encrypted.
ehWriteConf = {
    'eventhubs.connectionString' : sc._jvm.org.apache.spark.eventhubs.Event-
HubsUtils.encrypt(connection_string)
}

checkpointPath = userhome + "/event-hub/write-checkpoint"
dbutils.fs.rm(checkpointPath, True)

(activityStreamDF
    .writeStream
    .format("eventhubs")
    .options(**ehWriteConf)
    .option("checkpointLocation", checkpointPath)
    .start())
```

Event Hubs configuration

Assemble the following:

- A `startingEventPosition` as a JSON string
- An `EventHubsConf`
 - to include a string with connection credentials
 - to set a starting position for the stream read
 - to throttle Event Hubs' processing of the streams

```
%python

import json

# Create the starting position Dictionary
startingEventPosition = {
    "offset": "-1",
    "seqNo": -1,           # not in use
    "enqueuedTime": None,  # not in use
    "isInclusive": True
}

eventHubsConf = {
    "eventhubs.connectionString" : sc._jvm.org.apache.spark.eventhubs.Event-
```

```
tHubsUtils.encrypt(connection_string),  
    "eventhubs.startingPosition" : json.dumps(startingEventPosition),  
    "setMaxEventsPerTrigger": 100  
}
```

READ stream using Event Hubs

The `readStream` method is a **transformation** that outputs a DataFrame with specific schema specified by `.schema()`.

```
%python  
  
from pyspark.sql.functions import col  
  
spark.conf.set("spark.sql.shuffle.partitions", sc.defaultParallelism)  
  
eventStreamDF = (spark.readStream  
    .format("eventhubs")  
    .options(**eventHubsConf)  
    .load()  
)  
  
eventStreamDF.printSchema()
```

Most of the fields in this response are metadata describing the state of the Event Hubs stream. We are specifically interested in the `body` field, which contains our JSON payload.

Noting that it's encoded as binary, as we select it, we'll cast it to a string:

```
%python  
bodyDF = eventStreamDF.select(col("body").cast("STRING"))
```

Each line of the streaming data becomes a row in the DataFrame once an **action** such as `writeStream` is invoked.

Notice that nothing happens until you engage an action, i.e. a `display()` or `writeStream`.

```
%python  
display(bodyDF, streamName= "bodyDF")
```

While we can see our JSON data now that it's cast to string type, we can't directly manipulate it.

Parse the JSON payload

The Event Hub acts as a sort of "firehose" (or asynchronous buffer) and displays raw data in the JSON format. If desired, we could save this as raw bytes or strings and parse these records further downstream in our processing. Here, we'll directly parse our data so we can interact with the fields.

The first step is to define the schema for the JSON payload.

Both time fields are encoded as `LongType` here because of non-standard formatting.

```
%python

from pyspark.sql.types import StructField, StructType, StringType, LongType, DoubleType

schema = StructType([
    StructField("Arrival_Time", LongType(), True),
    StructField("Creation_Time", LongType(), True),
    StructField("Device", StringType(), True),
    StructField("Index", LongType(), True),
    StructField("Model", StringType(), True),
    StructField("User", StringType(), True),
    StructField("gt", StringType(), True),
    StructField("x", DoubleType(), True),
    StructField("y", DoubleType(), True),
    StructField("z", DoubleType(), True),
    StructField("geolocation", StructType([
        StructField("PostalCode", StringType(), True),
        StructField("StateProvince", StringType(), True),
        StructField("city", StringType(), True),
        StructField("country", StringType(), True)
    ]), True),
    StructField("id", StringType(), True)
])
```

Parse the data

Next we can use the function `from_json` to parse out the full message with the schema specified above.

When parsing a value from JSON, we end up with a single column containing a complex object:

```
%python

from pyspark.sql.functions import col, from_json

parsedEventsDF = bodyDF.select(
    from_json(col("body"), schema).alias("json"))

parsedEventsDF.printSchema()
```

Here's the output:

```
root
|-- json: struct (nullable = true)
|   |-- Arrival_Time: long (nullable = true)
|   |-- Creation_Time: long (nullable = true)
|   |-- Device: string (nullable = true)
|   |-- Index: long (nullable = true)
|   |-- Model: string (nullable = true)
|   |-- User: string (nullable = true)
|   |-- gt: string (nullable = true)
```

```
|   |-- x: double (nullable = true)
|   |-- y: double (nullable = true)
|   |-- z: double (nullable = true)
|   |-- geolocation: struct (nullable = true)
|   |   |-- PostalCode: string (nullable = true)
|   |   |-- StateProvince: string (nullable = true)
|   |   |-- city: string (nullable = true)
|   |   |-- country: string (nullable = true)
|   |-- id: string (nullable = true)
```

Note that we can further parse this to flatten the schema entirely and properly cast our time fields.

```
%python

from pyspark.sql.functions import from_unixtime

flatSchemaDF = (parsedEventsDF
    .select(from_unixtime(col("json.Arrival_Time")/1000).alias("Arrival_"
    Time").cast("timestamp"),
    (col("json.Creation_Time")/1E9).alias("Creation_Time").cast("-
    timestamp"),
    col("json.Device").alias("Device"),
    col("json.Index").alias("Index"),
    col("json.Model").alias("Model"),
    col("json.User").alias("User"),
    col("json.gt").alias("gt"),
    col("json.x").alias("x"),
    col("json.y").alias("y"),
    col("json.z").alias("z"),
    col("json.id").alias("id"),
    col("json.geolocation.country").alias("country"),
    col("json.geolocation.city").alias("city"),
    col("json.geolocation.PostalCode").alias("PostalCode"),
    col("json.geolocation.StateProvince").alias("StateProvince"))
)
```

This flat schema provides us the ability to view each nested field as a column.

Event Hubs FAQ

This [FAQ¹¹](#) can be an invaluable reference for occasional Spark-EventHub debugging.

Continue to the next step

After you've completed the notebook, return to this screen, and continue to the next step.

¹¹ <https://github.com/Azure/azure-event-hubs-spark/blob/master/FAQ.md>

Knowledge check

Question 1

When doing a write stream command, what does the `outputMode ("append")` option do?

- The append mode allows records to be updated and changed in place
- The append outputMode allows records to be added to the output sink
- The append mode replaces existing records and updates aggregates

Question 2

In Spark Structured Streaming, what method should be used to read streaming data into a DataFrame?

- `spark.readStream`
- `spark.read`
- `spark.stream.read`

Question 3

What happens if the `command` option (`"checkpointLocation"`, pointer-to-checkpoint directory) is not specified?

- It will not be possible to create more than one streaming query that uses the same streaming source since they will conflict
- The streaming job will function as expected since the `checkpointLocation` option does not exist
- When the streaming job stops, all state around the streaming job is lost, and upon restart, the job must start from scratch

Summary

Apache Spark Structured Streaming enables you to process streaming data and perform analytics in real time. You can use Azure Event Hubs together with Structured Streaming to process and analyze messages in real time.

Now that you have concluded this module, you should know:

- The key features and uses of Structured Streaming.
- How to stream data from a file and write it out to a distributed file system.
- How to list and stop active streams.
- How to use sliding windows to aggregate over chunks of data rather than all data.
- How to apply watermarking to throw away stale old data that you do not have space to keep.
- How to plot live graphs using `display`.
- How to connect to Event Hubs and write a stream to your event hub.
- How to read a stream from your event hub.
- How to define a schema for the JSON payload and parse the data do display it within a table.

Clean up

If you plan on completing other Azure Databricks modules, don't delete your Azure Databricks instance yet. You can use the same environment for the other modules.

Delete the Azure Databricks instance

1. Navigate to the Azure portal.
2. Navigate to the resource group that contains your Azure Databricks instance.
3. Select **Delete resource group**.
4. Type the name of the resource group in the confirmation text box.
5. Select **Delete**.

Module Lab information

Lab 11 - Create a Stream Processing Solution with Event Hubs and Azure Databricks

- **Estimated Time:** 50 - 60 minutes
- **Lab files:** The files for this lab are located in the *Instructions\Labs\15* folder.

Lab overview

In this lab, students will learn how to ingest and process streaming data at scale with Event Hubs and Spark Structured Streaming in Azure Databricks. The student will learn the key features and uses of Structured Streaming. The student will implement sliding windows to aggregate over chunks of data and apply watermarking to remove stale data. Finally, the student will connect to Event Hubs to read and write streams.

Lab objectives

After completing this lab, you will be able to:

- Understand the key features and uses of Structured Streaming
- Stream data from a file and write it out to a distributed file system
- Use sliding windows to aggregate over chunks of data rather than all data
- Apply watermarking to remove stale data
- Connect to Event Hubs read and write streams

Module summary

module-summary

In this module the student have explored how Structured Streaming helps to process streaming data in real time using Azure Databricks and Azure Event Hubs. The student has been enabled and taught how to aggregate data over windows of time.

Learning objectives

In this module, you have learned to:

- Understand the key features and uses of Structured Streaming.
- Stream data from a file and write it out to a distributed file system.
- Use sliding windows to aggregate over chunks of data rather than all data.
- Apply watermarking to throw away stale old data that you do not have space to keep.
- Connect to Event Hubs read and write streams.

Post Course Review

After the course, consider visiting the website that explores **structured streaming**¹² patterns with Azure Databricks and Event Hubs, where the associated documentation goes into more depth about this pattern.

Important

Remember

Should you be working in your own subscription while running through this content, it is best practice at the end of a project to identify whether or not you still need the resources you created. Resources left running can cost you money.

You can delete resources one by one, or just delete the resource group to get rid of the entire set.

¹² <https://docs.microsoft.com/en-us/azure/databricks/spark/latest/structured-streaming/>

Answers

Question 1

When doing a write stream command, what does the `outputMode ("append")` option do?

- The append mode allows records to be updated and changed in place
- The append outputMode allows records to be added to the output sink
- The append mode replaces existing records and updates aggregates

Explanation

The outputMode "append" option informs the write stream to add only new records to the output sink. The "complete" option is to rewrite the full output - applicable to aggregations operations. Finally, the "update" option is for updating changed records in place.

Question 2

In Spark Structured Streaming, what method should be used to read streaming data into a DataFrame?

- `spark.readStream`
- `spark.read`
- `spark.stream.read`

Explanation

Use the `spark.readStream` method to start reading data from a streaming query into a DataFrame.

Question 3

What happens if the command `option("checkpointLocation", pointer-to-checkpoint directory)` is not specified?

- It will not be possible to create more than one streaming query that uses the same streaming source since they will conflict
- The streaming job will function as expected since the `checkpointLocation` option does not exist
- When the streaming job stops, all state around the streaming job is lost, and upon restart, the job must start from scratch

Explanation

Setting the `checkpointLocation` is required for many sinks used in Structured Streaming. For those sinks where this setting is optional, keep in mind that when you do not set this value, you risk losing your place in the stream.