Azure Data Engineer Certification (Book)

Metadata

- Title: Azure Data Engineer Associate Certification Guide
- Author: Newton Alex
- Reference: Amazon

Table of Contents

Notes

These notes will focus on theory over application. Reference GitHub Repository for code examples: [Azure Data Engineer DP203](#).

Chapter 1: Introducing Azure Basics

Azure provides several cloud, hybrid, and on-premises services such as VMs, networks, compute, databases, messaging, ML, AI, IoT, and many more services while focusing on security and compliance.

- Advantages of cloud-based computing
  - Geo-replication
  - High availability
  - Data redundancy
  - Scalability
  - Elasticity

Exploring Azure

Azure Accounts

Azure Account > Subscriptions > Resources Groups > Resources

- An Azure account refers to the Azure Billing account
- It is mapped to the email used to sign up for Azure
- Accounts can contain multiple subscriptions
- Subscriptions can contain multiple resource groups
  - Billing is computed at the subscription level
- Resource groups can contain multiple recourses

Azure Subscriptions

- A subscription is a container for all resources created under that subscriptions
- Contains the details of all the services (VMs, networks, storage, etc.) used during a month that will be used for billing purposes

- Subscriptions can be created to split billing into logical divisions bases on use case
    - Team division (sales, marketing, finance, etc.)
    - Region division (East US, Central US, Asia Pacific, etc.)

Resource Groups and Resources

- Resource groups are logical groups of resources belonging to an application or team
- Acts as a tag associated with a collections of resources for easy querying, monitoring, and managing
- All resources within a resource group can be deleted at once by deleting the resource group
    - Resources refers to VMs, stores, databases, functions (etc.) that can be created in Azure

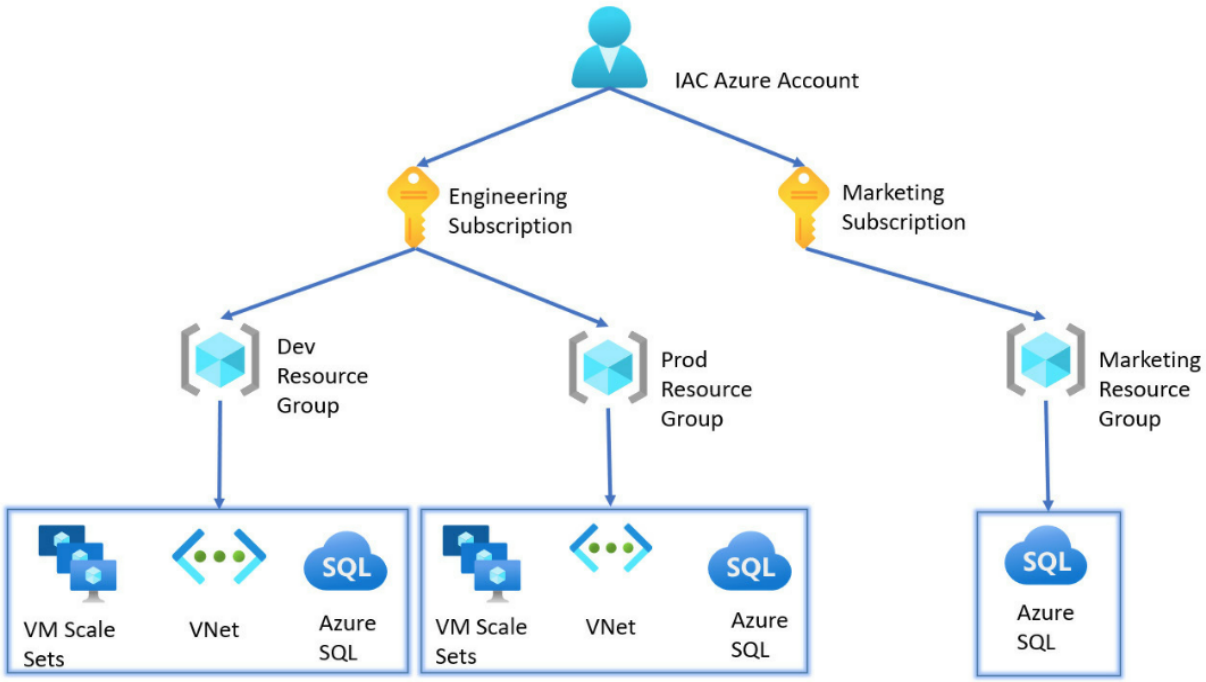


Figure 1.2 – Relationship between accounts, subscriptions, resource groups, and resources

Exploring Azure Services

- Azure services are categorized into;
    - Infrastructure as a Service (IaaS)

        - Bare infrastructure such as VMs, VNets, and storage
        - The rest of the application stack needs to be build
        - Gives developers the most flexibility in terms of OS versions, library versions, custom patches, etc.
    - Platform as a Service (PaaS)
        - Software platforms are pre-installed and pre-configured by Azure
        - Some level of tuning is available, but this lacks the flexibility to choose particular versions, patching, etc.
    - Software as a Service (SaaS)
        - Named Functions as a Service (FaaS) by Azure
        - Developers don't get to see any software installation details
        - Usually comes with a notebook-like interface or an API interface for submitting jobs
        - Azure handles instantiating the service, scaling the service, and running jobs
        - Easiest and quickest to implement, but most restrictive in terms of custom tuning

Exploring Azure VMs

- VMs are software abstractions of physical hardware
    - Multiple VMs can run on a single machine with a portion of the host machines resources allocated to each VM
    - Machine resources include CPU, memory, and storage

- Azure VM advantages
    - Faster deployments
    - Scalability
    - Security isolation
    - Elasticity

- Azure Managed Disks
    - Virtual hard disks mounted to an Azure VM
    - Completely managed by Azure (no need to worry about OS upgrades, security patches, etc.)

    - Over 99.999% availability by storing three different copies of data on different servers
    - Can be allocated to availability sets and availability zones (distributed across racks and data centers)
    - Options provided for data encryption at rest and disk-level encryptions

Exploring Azure Storage

Azure Blob Storage

- Stores unstructured data (videos, audio, metadata, log files, text, binary, etc.)
- Provides flat storage directory
    - Account > Container > File(s)
    - Does not allow containers within containers
- Highly scalable and cost-effective storage solution
- Provides support for tiered storage based on access patterns and usage frequency (hot tiers, cold tiers, archived)
- Data can be accessed via REST endpoints, as well as client libraries available in a wide set of languages
- Supports role-based access controls (RBAC)

Azure Data Lake Gen 2 (ADLS)

- Preferred options for data lake solutions on Azure
- Optimized for big data analytics
- Provides hierarchal namespace support on top of blob storage (directories are supported)
    - Account > Container > Container(s) > File(s)
- Supports role-based access controls (RBAC)
- Hadoop-compatible filesystem (allows building of open-source analytics pipelines)
- When creating a blob storage account, simply check the "Enable hierarchal namespace" option to create a Data Lake Gen2 account

Azure Files

- Provides remote file shares that can be mounted using Server Message Block (SMB) or Network File Share (NFS) protocols

- Great storage option for migrating on-premises workloads to the cloud with a lift and shift model
- Particularly useful for cases that need shared data, shared configurations, shared applications, and more across multiple users, teams or regions

Azure Queues

- Used to store a large number of messages that can be accessed asynchronously between the source and destination
- Helps decouple applications so they can scale independently
- Two types of queues are Storage Queues and Service Bus
    - Storage Queue
        - Simple message processing
        - Stores up 500TB per account
        - Each message can be up to 64KB
    - Service Bus
        - Advanced message processing (pub-sub, models, strict ordering, blocking and non-blocking APIs)
        - Stores up to 80 GB per account
        - Each message can be up to 1MB

Azure Tables

- Key-Value stores provided by Azure
- Good for storing structured non-relational data (NoSQL)
- Two solutions available are Azure Table Storage and Cosmos DB
- Both provide the same table model and CRUD features (Create, Read, Update, and Delete)
- Difference between both solutions is scalability (Cosmos DB is the premium version and will be discussed later)
    - Cosmos DB is the preferred NoSQL store, but is no longer in the exam

Exploring Azure Networking (VNet)

- A VNet securely ties all resources (VMs, stores, databases, etc.) together in a private network
- Used to encapsulate cloud or on-premises services within a secure boundary

- Restricts who can access services and from which endpoints
- Azure Networking provides the following four main services;
    - Secure connectivity within Azure resources using the basic VNet, VNet Peering, and Service Endpoints
    - Networking beyond the Azure Cloud and into the internet and hybrid clouds using Express Routers, Private Endpoints, and Point-to-Site and Site-to-Site VPNs
    - Network filtering (firewall rules) that can be implemented either via the Network or App Security Groups
    - Network routing abilities that allow us to configure network routes using Route Tables and Border Gateway Protocols

Exploring Azure Compute

- Azure Compute is a generic term for all the compute-focused technologies in Azure
- Common compute services provided by Azure
    - VM Scale Sets
    - Azure App Service
    - Azure Kubernetes Service (AKS)
    - Azure Functions
    - Azure Service Fabric

VM Scale Sets

- Collection of load-balanced VMs that can be used to build highly scalable services

Azure App Service

- Allows us to develop and host web apps, mobile apps, and APIs using a wide selection of languages
- Fully managed services that provide support for the entire development life cycle (development, CI/CD, releases, maintenance, debugging, and scaling)

Azure Kubernetes Service (AKS)

- Life cycle management for containerized apps

- Supports docker images

Azure Functions

- Use functions to write processing logic based on event triggers and bindings such as a transaction in a database, an IoT event, or a REST call

Azure Service Fabric

- Powerful cluster technology that takes care of app deployment, scaling, upgrades, and maintenance for microservice-based applications
- Similar to AKS but for non-containerized applications

Azure Batch

- Used to run large parallel processing applications or high-performance computing applications
- Provides the necessary resource management, scheduling, and scaling support to run any traditional MPP programs
- Spins up the VMs and deploys and runs ours programs in a parallel manner
- Can dynamically scale up and down as required to optimize the cost
- Can be used for high volume batch processing, financial modeling, video rendering, weather prediction model generation, etc.

Summary

With that, we have completed our first chapter. If it was too overwhelming for you, don't worry – this chapter was just meant to provide an overview of Azure. By the time you complete the next few chapters, your confidence will increase. On the other hand, if this chapter was easy for you, then you are probably already aware of some level of cloud technologies, and the next set of chapters should also be easy for you.

Now that you have completed this chapter, you should know how to navigate the Azure portal. You now understand the relationship between Azure accounts, subscriptions, resource groups, and resources. You also know how to create new VMs, Storage instances, VNets, and so on using both the Azure portal and the CLI. You are also aware of the major compute services that are available in Azure. With this foundational

knowledge in place, we can move on to more interesting and certification-oriented topics.

We will be exploring Azure storage technologies in the next chapter.

Chapter 2: Designing a Data Storage Structure

- Azure storage technologies include;
  - Azure Storage
    - Blobs
    - Files
    - Queues
    - Tables
  - Azure Data Lake Gen2
  - Azure SQL Database
  - Azure Dedicated SQL Pools (Formerly SQL DW)
  - Azure Synapse SQL Warehouse
    - SQL Pools (Dedicated)
    - Spark Pools
  - Azure Cosmos DB

Designing an Azure Data Lake

- Data Lakes are distributed data stores that hold large volumes of diverse data at a low cost
- Capable of storing structured, semi-structured, unstructured, and streaming data
- Data Lake solutions usually encompass the following layers;
  - Storage
  - Compute (can include batch or stream ETL processing)
  - Serving

How a Data Lake Differs from a Data Warehouse

- Data warehouses primarily store structured data

- Data Lakes usually act as a landing zone for different sources and types of data
    - This raw data is processed and eventually loaded into structured data stores such as a data warehouse

When to Use a Data Lake

- When data is too big for structured storage systems
- Storing raw data that needs to be stored for further processing
- Storing continuous data (streaming)
- Storing semi-structured or unstructured data
- Storing processed data for advanced tasks (ML/AI)
- As a data staging layer

Data Lake Zones

- Landing Zone (Raw Zone): Where raw data is stored after ingestion
- Transformation Zone: Where batch or stream processing takes place
- Serving Zone: Where curated data is stored to serve BI tools (data usually adheres to well-defined schemas)

Example Data Lake Architecture (Azure Services)

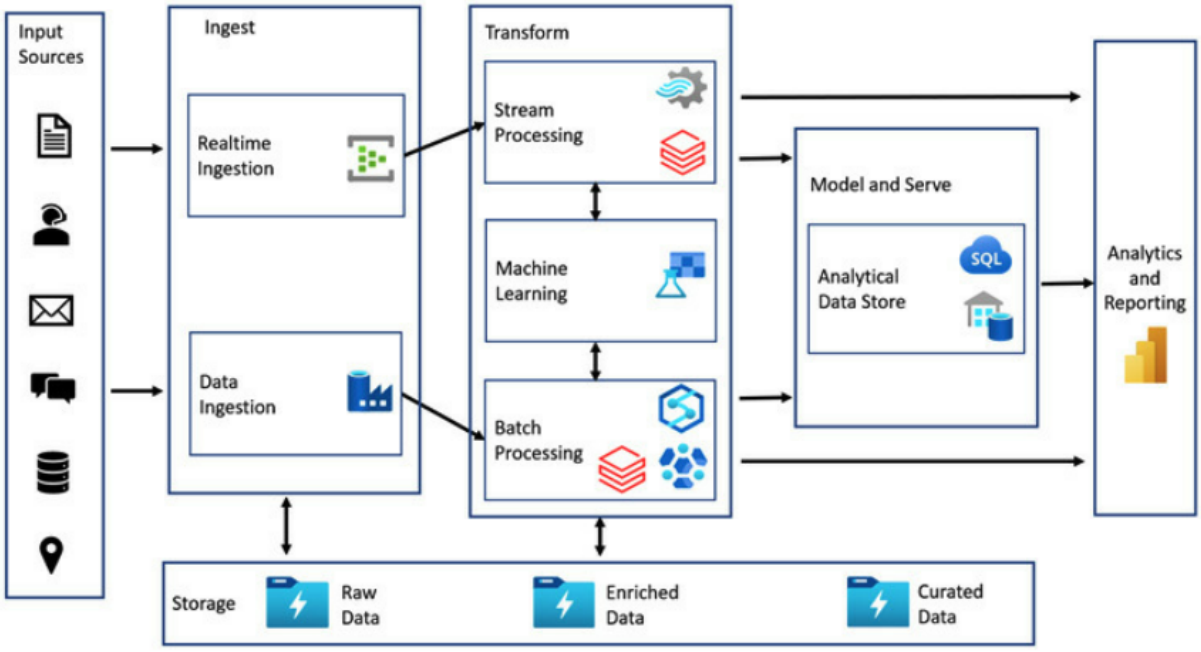


Figure 2.1 – Data lake architecture using Azure services

Batch Processing

- Data usually lands in a Landing Zone from various input sources
- Once landed, orchestration frameworks trigger data processing pipelines
    - Cleansing, filtering, aggregating, curated data for BI
- Azure Data Factory is a service used to ingest and build pipelines

Stream Processing

- Refers to near real-time processing of data
- Processes data as and when it arrives
- Typically, data arrives in small files in quick succession (often referred to as messages)
- The streaming system will apply checks for data formats, processes the data, and writes the data to a store
- Stores used for stream processing need to support high-column writes with low latency

Lambda Architecture

- Hybrid architecture that uses a combination of fast and slow processing pipelines
- The slow path processes a larger volume of data and produces accurate results at the expense of time (batch)
- The fast path processes smaller data sets (usually sampled) and gives an approximate result much quicker (stream)
- Both of these pipelines feed into a serving layer that updates the incremental updates from the fast path into the baseline data from the slow path

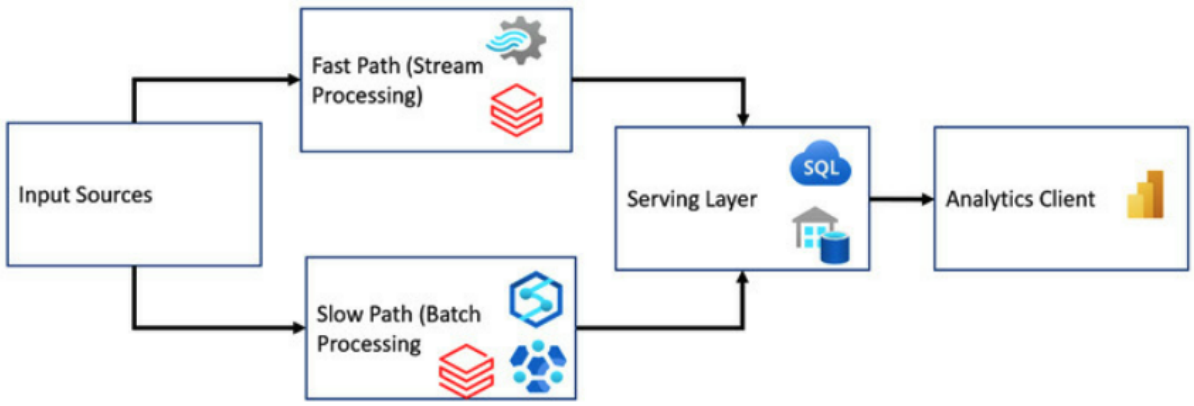


Figure 2.5 – Lambda architecture

Kappa Architecture

- Alternative to Lambda architecture that focuses only on the fast path
- Reduces complexity and avoids dual processing
- Often used for real-time ML and applications where baseline data does not change often
- A long-term store is used for historical data if needed for recalculations

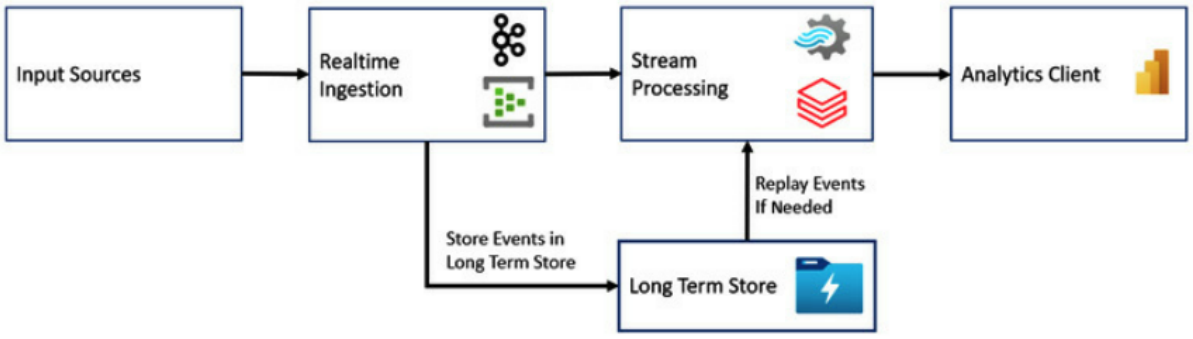


Figure 2.6 – Kappa architecture

Azure Technologies Used to Build Data Lakes

- ADLS Gen2 or Blob Storage
  - Blob storage is suitable if requirements mostly center around storing unstructured data without the need of access-control lists (ACLs)
- Azure Data Factory (ADF)
  - Data ingestion
  - Data transformation
  - Orchestration
- Azure Synapse Analytics
  - Complete suite of integrated analytical services (all in one workspace)
    - Azure Stores
    - Cosmos DB (Specialized NoSQL Store)
    - SQL Data Warehouse
    - Synapse Spark
    - Azure Data Factory
    - Azure Active Directory (AAD)
    - Azure Purview (Governance)
    - Identity and Access Management (IAM)

- Azure Databricks
  - Notebook experience
  - Integration with other Azure services
  - Used to write Spark for data processing
- Azure HDInsight
  - Open-source versions of Apache Spark, Apache Hive, and Apache HBase
  - Used to write Spark for data processing
- Azure Stream Analytics
  - Stream processing
  - Integrates with all Azure storage services and can directly connect to Power BI
- Power BI
  - Tools that can operate on big data to provide visualizations and insights
  - Build-in connectivity with Azure services
- Azure ML
  - Workspace for advanced analytics and prediction
  - Provides support for a wide range of algorithms and models for data analytics
  - Options for low-code and code development

Selecting the Right File Types for Storage

- Data often arrives in text files, log files, CSV, JSON, XML, etc
  - Not the best formats for data analytics
- Big data recommends three file formats: Avro, Parquet, Optimized Row Columnar (ORC)
- Data Lake supports the use of multiple file formats in a storage account (polyglot persistence)
- Main considerations when choosing a file format;
  - Type of workload
  - Storage cost (file formats that support better compression become important)
  - Compression vs performance

Avro

- Row based storage format
  - Each complete row is stored one after the other in file storage

- Good for write intensive transactional workloads such as ETL jobs that need to read entire files

Parquet

- Column-based storage format
  - Stores data from each related column together one after the other in a file
- Good for read intensive jobs such as analytical workloads
- Supports predicate pushdowns for query optimization

Optimized Row Columnar (ORC)

- Column-based storage format similar to Parquet files
- Supports ACID transactions and predicate pushdowns
  - Atomicity
  - Consistency
  - Isolation
  - Durability

Suggested File Formats per Storage Zone

- Landing Zone or Raw Zone
  - Avro is best for large data file that require compression, and for a high number of ETL jobs
  - Parquet is a good compromise between compression and performance if one file format is preferred
- Transformation Zone
  - ORC works best with Hive
  - Parquet works best with Spark

Choosing the Right File Types for Analytical Queries

- Any format that supports fast reads is best for analytics workloads
- Parquet and ORC are the best file formats for analytics queries
  - Choose ORC for Hive or Presto based workloads
  - Choose Parquet for Spark or Drill-based workloads

Designing Storage for Efficient Querying

- Two types of querying services on Azure
  - Azure SQL, Synapse Serverless/Dedicated SQL Pools
  - Analytical Engines Spark/Hive
- Techniques available for efficient querying can be grouped in the following three layers;
  - Storage Layer: Partitioning, Data Pruning, and Eventual Consistency
  - Application Layer: Data Caching and Application Tuning
  - Query Layer: Indexing and Materialized Views

Storage Layer

Partitions

- Partitions refer to the way we split data
- Data is divided evenly and related data is grouped together in the same partitions
- Partition strategies are difficult to change so its important to design correctly
- Frequently accessed data (lookup or catalog data) can be replicated across partitions
- Reducing Cross-Partition Operations and Joins
  - Minimize cross-partition joins by running jobs in parallel within each partition
  - Aggregate only the final results

Data Pruning

- Process of ignoring unnecessary data
- Reduces input/output (I/O) operations

Eventual Consistency

- Consistency refers to how soon internal copies of data will reflect changes made to primary copy
- There are typically two level of consistency
  - Strong Consistency: Ensures all data copies are updated before the user can perform any other operation
  - Eventual Consistency: Data in each of the copies gets updated gradually over time
    - Tends to be faster

- Best case if the business can tolerate eventual consistency

Application Layer

Optimizations deal with how efficiently we use the core components (CPU, memory, network, etc)

Tuning Applications

- Spark and Hive can be optimized by configuring the number or parallel executions, memory, CPU, networking, bandwidth, and the size of containers
- Select VMs that have enough memory, CPU, disk, and network bandwidth for our application

Caching

- Refers to storing intermediate data in faster storage layers to speed up queries

Query Layer

Indexing

- Indexing is one of the most important techniques to use for efficient querying
- Primary and secondary indexes are based on certain key columns
- This prevents complete table scans when querying data which speeds up query runtime

Materialized Views

- Views are logical projections of data from multiple tables
- Materialized views are pre-populated versions of views

Designing Storage for Data Pruning

- Data Pruning refers to pruning or snipping unnecessary data so that queries do not need to read the entire input dataset
- Reducing the amount of data read naturally improves query performance
- If we organize input data into physical folders that correspond to partitions, we can skip reading entire folders that are not needed

SQL Data Pruning

- RANGE RIGHT means the values specified in the PARTITION syntax each belong to the right side of the range
    - Partition 1: All Dates < 20220101
    - Partition 2: 20220101 to 20220131 (January)
    - Partition 3: 20220201 to 20220228 (February)
    - Partition 4: 20220301 to 20220331 (March)

Spark Data Pruning

- Write the files to different partitions (folder directories)
- Partitioning data with good folder structures can help improve efficiency of queries

Designing Folder Structures for Data Transformations (Data Lake)

- Important to design a flexible and maintainable folder structure
- Points to keep in mind
    - Human readability
    - Representation of organizational structure
    - Distinguish sensitive data
    - Manageability of ACLs
    - Optimize for faster and evenly distributed reads
    - Consider subscription limits

Streaming and IoT Scenarios

- Complicated as there will be a lot of files streaming from different sources
- Security policies can be provided at a directory level
- Folder structure recommended by Microsoft
    - Region / Subject Matter(s) / YYYY / MM / DD / HH
- Folder structure recommended by Microsoft for sensitive data use cases
    - Region / General / Subject Matter(s) / YYYY / MM / DD / HH
    - Region / Sensitive / Subject Matter(s) / YYYY / MM / DD / HH

Batch Scenarios

- In batch processing systems we read input data and write processed data into output directories
- We should maintain separate directories for inputs and outputs, as well as a directory for bad data (when issues arise)
    - Region / Subject Matter(s) / IN / YYYY / MM / DD / HH
    - Region / Subject Matter(s) / OUT / YYYY / MM / DD / HH
    - Region / Subject Matter(s) / BAD / YYYY / MM / DD / HH
    - We can also add a General and Sensitive directory if appropriate
- Do not create directories starting with date folders, as that will make ACL (security) applications more tedious.

Designing a Distribution Strategy

- Distribution strategies are techniques used in Synapse Dedicated SQL Pools
- Synapse Dedicated SQL Pools are massively-parallel processing (MPP) systems that split queries into 60 parallel queries before execution
- These smaller queries run on a "distribution"
    - A distribution is a basic unit of processing and storage for a dedicated SQL Pool
- Dedicated SQL Pools use Azure Storage to store data and provides three different ways to distribute the data among distributions
    - Round-robin tables
    - Hash tables
    - Replicated tables
- We should base our distribution strategy on business requirements

Round-Robin Tables

- Data is serially distributed among all distributions
- This is the default distribution type when one is not specified
- Quickest option for loading data, but not optimized for queries that include joins
- Best used for staging data or temporary tables where data is mostly read

Hash Tables

- Rows are distributed to different distributions based on a hash key
- Hash key is usually a column in the table

- Best used for tables that are queried with joins and aggregations

Replicated Tables

- Entire table data is copied over to all distributions
- Best used for small tables such as quick lookup tables (LUTs)

Designing a Data Archiving Solution

- Azure provides three storage tiers;
  - Hot Access Tier
  - Cold Access Tier
  - Archive Access Tier

Hot Access Tier

- Ideal for data that is accessed frequently
- Provides the lowest access cost, at a higher storage cost

Cold Access Tier

- Ideal for data that is accessed occasionally such as data used for backups or monthly reports
- Provides lower storage cost, at a higher access cost
- Data in a cold tier should be stored for at least 30 days (early deletion or tier change could result in a penalty)

Archive Access Tier

- Ideal for storing data for long durations such as compliance, long-term backups, archive data
- Offline storage solution (unable to access data unless it is rehydrated to an online tier)
- Provides the lowest storage cost, with no access cost since it is inaccessible without moving to a hot or cold tier
- Data in a archive tier should be stored for at least 180 days (early deletion or tier change could result in a penalty)

Data Life Cycle Management

- Azure blob storage provides tools for data life cycle management
- Policies can be defined such as;
    - How long data needs to be stored in a hot tier
    - When to move data to a different tier
    - When to delete blobs
- Policies can be defined in the Azure Portal
    - Navigate to the storage account
    - Click on Data Management > Life Cycle Management
    - Click + to add a new rule
- Azure runs data life cycle policies once a day so it could take up to 24 hours for new policies to be applied

Summary

With that, we have come to the end of our second chapter. We explored the various data lake designs in detail and learned good practices for designing one. You should now be comfortable answering questions related to data lake architectures and the storage, compute, and other technologies involved in creating a data lake. You should also be familiar with common file formats such as Avro, Parquet, and ORC and know when to choose which file formats. We also explored different optimization techniques such as data pruning, partitioning, caching, indexing, and more. We also learned about folder structures, data distribution, and finally, designing a data life cycle by using policies to archive or delete data. This covers the syllabus for DP-203 exam, Design a Data Storage Structure chapter. We will be reinforcing the learnings from this chapter via implementation details and tips in the following chapters.

Let's explore the concept of partitioning in more detail in the next chapter.

Chapter 3: Designing a Partition Strategy

Data partitioning refers to dividing data and storing it in physically different locations

Understanding the Basics of Partitioning

Benefits of Partitioning

- Performance
- Scalability
- Manageability
- Security

Improving Performance

- Partitioning improves parallelization of queries by splitting data into smaller chunks
- Data pruning improves query performance byb ignoring non-relevant partitions
- Partitioning also improves ease of data archiving or deletion
    - In ADLS, if we partition data into YY / Month, we can easily archive or delete entire partitions based on age

Improving Scalability

- Two types of scaling
- Vertical scaling
    - Increasing capacity of individual machines by adding more memory, CPU, storage, or network
- Horizontal scaling
    - Increasing processing and storage capacity by adding more machines to a cluster
    - Data Lakes are based on the concept of horizontal scaling
    - Data partitioning helps naturally with horizontal scaling
        - By splitting partitions into individual nodes

Improving Manageability

- Partitioning reduces management overhead
- Applying access restrictions and data lifecycle activities at the partition level
    - Data lifecycle refers to archiving or deleting data based on age

Improving Security

- Partitioning can isolate secure data and apply independent access-control and audit rules to those partitions
- Allowing only privileged users to access secure data

Improving Availability

- Data is split into multiple partitions and stored in different machines
- Unaffected partitions can still be access if a machine(s) goes down

Designing a Partition Strategy for Files

Azure Blob Storage (Storage Account)

- Azure uses range partitioning for storing blobs
- Files that are related in sequence are stored together in a partition
- Azure Storage uses the following as the partition key
    - Account Name + Container Name + Blob Name
    - Blobs will continue to be stored int he same partition until it reaches the internal limit
    - When the limit is reached, Azure Storage repartitions and rebalances the data
    - No user intervention is needed as these partitioning strategies are implemented automatically

Azure Data Lake Gen 2

- Partitioning is handled through folder structure
- Folder structure recommended by Microsoft
    - Region / Subject Matter(s) / YYYY / MM / DD / HH
- Folder structure recommended by Microsoft for sensitive data use cases
    - Region / General / Subject Matter(s) / YYYY / MM / DD / HH
    - Region / Sensitive / Subject Matter(s) / YYYY / MM / DD / HH
- Security policies and data lifecycle management configurations can be added to individual folders

Designing a Partitioning Strategy for Analytical Workloads

- Three main types for analytical workloads

- Horizontal partitioning (sharding)
- Vertical partitioning
- Functional partitioning

Horizontal Partitioning

- Data is divided horizontally, subsets of rows are stored in different data stores (such as a database instance)
- Each subset (same schema as parent table) are called shards
- Each shard is stored in a different database instance

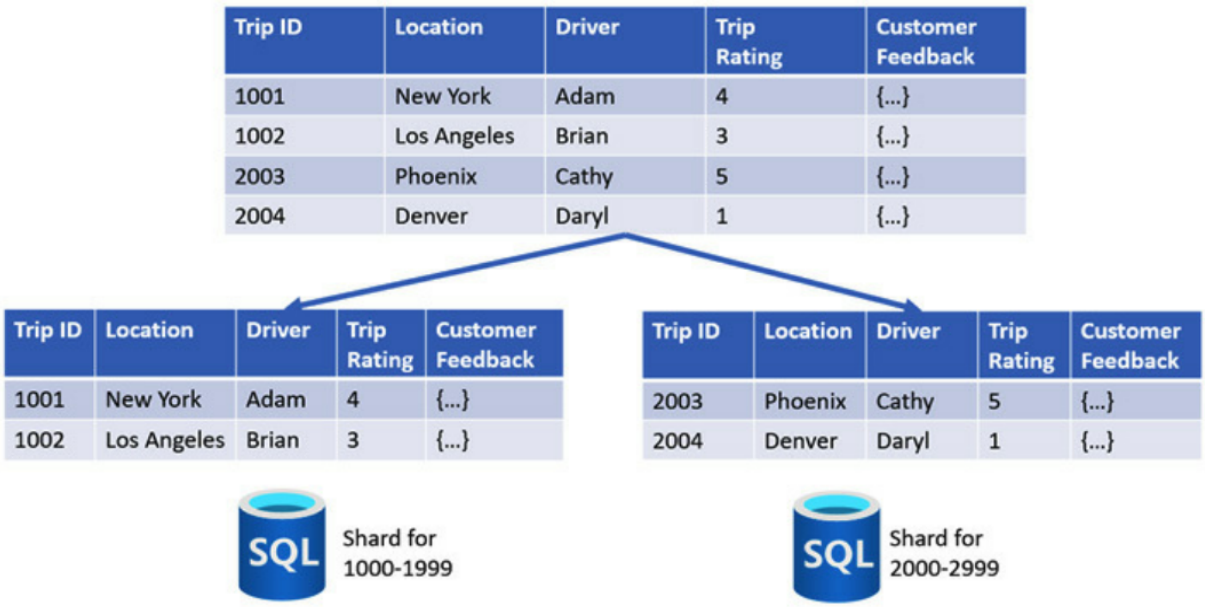


Figure 3.1 – Example of a horizontal partition

Selecting the Right Shard Key

- Very important to select the right shard key (partition key)
- Select a key that spreads the data so that application traffic to the partitions is evenly distributed
- Select a key that doesn't change often
- Select a key that generates hundreds of partitions (not tens or thousands)
- Be mindful that recent data is accessed more frequently and can become a bottleneck

Vertical Partitioning

- Data is divided vertically, subsets of columns are stored in different data stores (such as a database instance
- Ideal in cases where a subset of columns from a table are accessed frequently
- Similar to table normalization while being cognizant of frequently accessed attributes
    - In the example below, Trip Rating and Customer Feedback are likely less accessed than Location and Driver

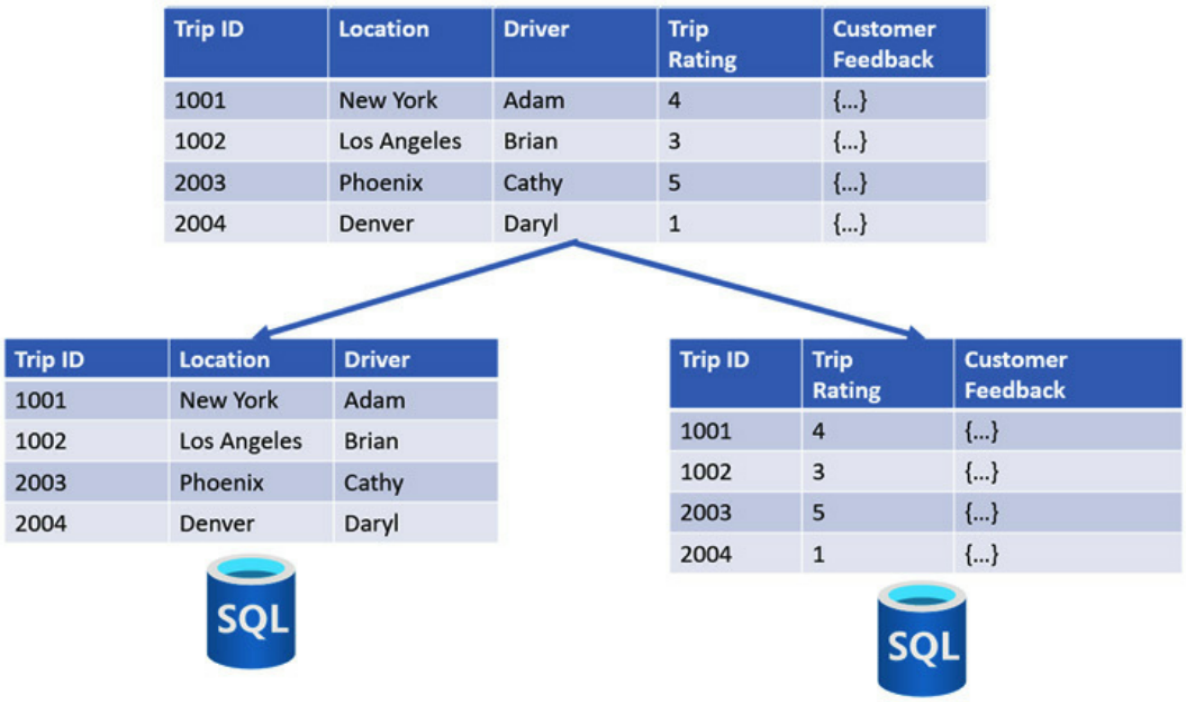


Figure 3.2 – Example of a vertical partition

Functional Partitioning

- Similar to vertical partitioning, except entire tables or entities are stored in different data stores (such as a database instance
- Used to segregate data belonging to
    - Different organizational domains
    - Frequently and infrequently used tables
    - Read-write and read-only tables
    - General data and sensitive data
- Helps apply different privacy and security rules for different partitions
- In the example below, customer data is moved into its own partitions

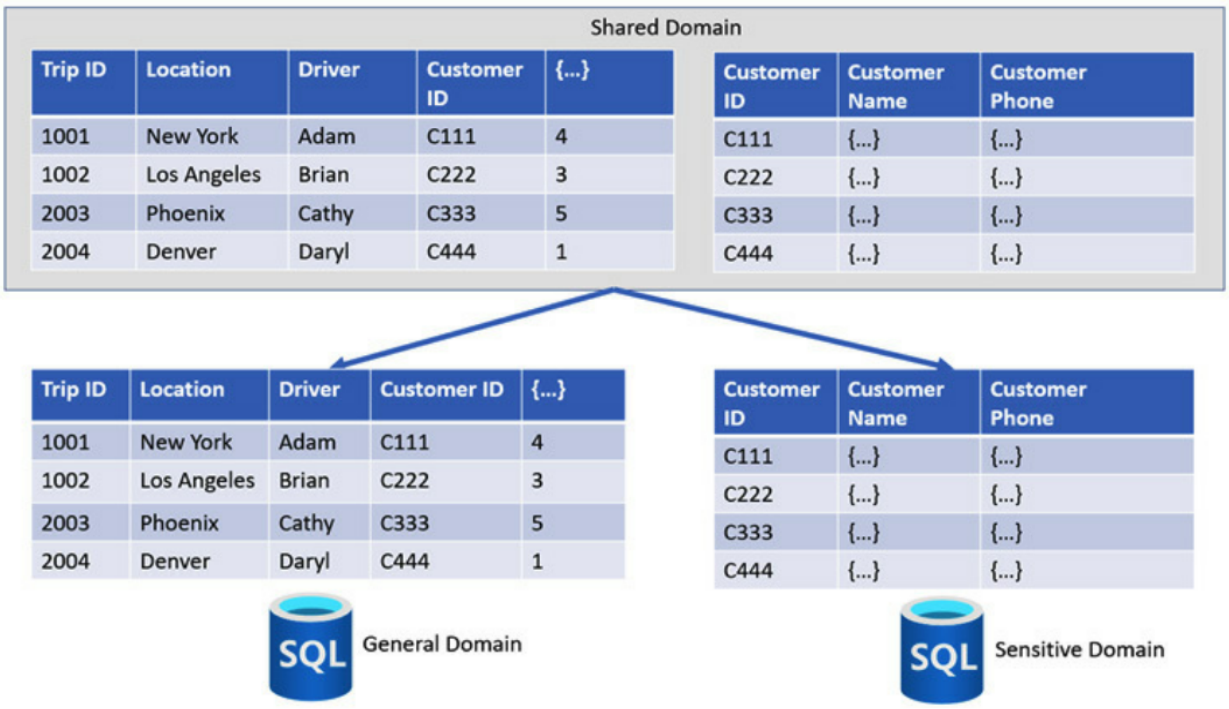


Figure 3.3 – Example of a functional partition

Designing a Partition Strategy for Efficiency and Performance

- Strategies to keep in mind when designing for efficiency and performance
  - Partition datasets into smaller chunks that can be run with optimal parallelism for multiple queries.
  - Partition the data such that queries don't end up requiring too much data from other partitions
  - Minimize cross-partition data transfers
  - Design effective folder structures to improve the efficiency of data reads and writes
  - Partition data such that a significant amount of data can be pruned while running queries
  - Partition in units of data that can be easily added, deleted, swapped, or archived (helps improve the efficiency of data life cycle management)

    - File sizes in the range of 256 megabytes (MB) to 100 gigabytes (GB) perform really well with analytical engines such as HDInsight and Azure Synapse (aggregate the files to these ranges before running the analytical engines on them)
    - For I/O-intensive jobs, try to keep the optimal I/O buffer sizes in the range of 4 to 16 MB; anything too big or too small will become inefficient
    - Run more containers or executors per virtual machine (VM) (such as Apache Spark executors or Apache Yet Another Resource Negotiator (YARN) containers)

Iterative Query Performance Improvement Process

- Identify business critical, frequently ran,a nd slow performing queries
- Analyze query plans for these queries using the EXPLAIN keyword
- Identify the joins and filters creating bottlenecks and their respective partitions
- Analyze the partitioning strategy for those partitions and make changes as needed

Designing a Partition Strategy for Azure Synapse Analytics (ASA)

- Synapse MPP Node-Based Architecture
    - Control Node
        - Brain of MPP architecture
        - Optimizes and coordinates parallel queries
        - Acts as the application or user front-end (handling connections and authorization)
    - Compute Node
        - Computational power
        - Distribution map
        - Distributes jobs across 1-60 compute nodes
        - Runs on DWU (Data Warehousing Unit)
    - Storage Node
        - Decoupled from compute nodes to keep unused data at rest
        - Data at rest only accumulates storage costs which are relatively cheap
- Synapse Analytics contains two compute engines
    - Dedicated SQL Pool (focus of this section)

    - Spark Pool (reviewed in the Spark Data Pruning section earlier)
- *Distribution Technique of a Synapse Dedicated SQL Pool:* *A dedicated SQL pool is a massively parallel processing (MPP) system that splits the queries into 60 parallel queries and executes them in parallel. Each of these smaller queries runs on something called a distribution. A distribution is a basic unit of processing and storage for a dedicated SQL pool. There are three different ways to distribute (shard) data among distributions, as listed here:*
    - *Round-robin tables*
    - *Hash tables*
    - *Replicated tables*
- Since data is already partitioned across 60 distributions, we need to be careful how we further partition the data

Performance Improvement While Loading Data

- Partitioning helps while loading data for queries in Dedicated SQL Pools
- Grouping data by timeframe makes it easier to use ADD or DELETE commands

Performance Improvement for Filtering Queries

- Partitioning can also help improve query performance by being able to filter data based on partitions (WHERE clause)

Table Types

- Apart from distribution types, ASA supports the following types of tables
    - Clustered colunstore
    - Clustered index
    - Heap

Identifying When Partitioning is Needed in ADLS Gen 2

- ADLS and Blob storage have various I/O bandwidth limits that are imposed at the Azure subscription level
- The below table shows a snapshot of Azure imposed storage limitations
- Ingress Rate: Rate at which we ingest data into an Azure Storage system
- Egress Rate: Rate at which we move data out of an Azure Storagesystem

Resource	Limit
Number of storage accounts per region per subscription, including standard, and premium storage accounts.	250
Maximum storage account capacity	5 PiB (can be increased by calling Azure Support)
Maximum request rate[1] per storage account	20,000 requests per second
Maximum ingress[1] per storage account (US, Europe regions)	10 Gbps
Maximum ingress[1] per storage account (regions other than US and Europe)	5 Gbps if RA-GRS/GRS is enabled, 10 Gbps for LRS/ZRS
Maximum egress for general-purpose v2 and Blob storage accounts (all regions)	50 Gbps
Maximum number of IP address rules per storage account	200
Maximum number of virtual network rules per storage account	200

Figure 3.4 – Some of the limits for Azure Blob storage as of the time this book was published

Summary

With that, we have come to the end of our third chapter. I hope you enjoyed learning about the different partitioning techniques available in Azure! We started with the basics of partitioning, where you learned about the benefits of partitioning; we then moved on to partitioning techniques for storage and analytical workloads. We explored the best practices to improve partitioning efficiency and performance. We understood the concept of distribution tables and how they impact the partitioning of Azure Synapse Analytics, and finally, we learned about storage limitations, which play an important role in deciding when to partition for ADLS Gen2. This covers the syllabus for the DP-203 exam, Designing a Partition Strategy. We will be reinforcing the learnings from this chapter via implementation details and tips in the following chapters.

Let's explore the serving layer in the next chapter.

Chapter 4: Designing the Serving Layer

- Data Lake storage design consists of three zones
  - Landing zone (raw, bronze)
  - Transformation zone (processed, silver)
    - Processed into a more useful format
  - Serving zone (curated, gold)
    - Derived data insights
- Landing and transformation zone focus on
  - Efficient storage
  - Processing large volume
  - Optimizing queries, etc
- Serving zone focuses on serving data efficiently
  - Often built using relational data stores (SQL based stored)
  - Relational stores can store data in more efficient normalized tables and perform queries faster
  - With Lakehouse architecture, the serving zone (gold) can stay in the Lake with Parquet files

Data Modeling and Schemas

- Data modeling is the process of designing how data will be represented in data stores
- Databases and data warehouses perform queries fast and efficiently
- When modeling the serving layer, we need to decide
  - Which SQL tables are required
  - What the relationships between tables will be
  - Which restrictions need to be imposed on these tables
- Dimensional modeling is most relevant to data warehousing

Dimensional Modeling

- Focuses on easier and faster information retrieval

- Most commonly used dimensional models are star and snowflake schemas

Designing Star and Snowflake Schemas

- Designing schemas refers to the process of designing various tables and the relationships among them
- Star schemas are used more frequently than snowflake schemas

Star Schemas

- Consists of two types of tables
  - Fact Tables: Quantitative tables
  - Dimension Tables: Descriptive or context tables
- Important points about star schemas
  - Fact tables usually have much higher volume
  - Dimension tables are not connected (independent of each other)
  - Data is not normalized (common to find replication in dimension tables)
  - Tables are designed for speed and ease of use (less joins required) resulting in faster queries
    - Optimized for BI queries
- In the below example, DriverID, CustomerID, DateID, and CabID are all primary keys

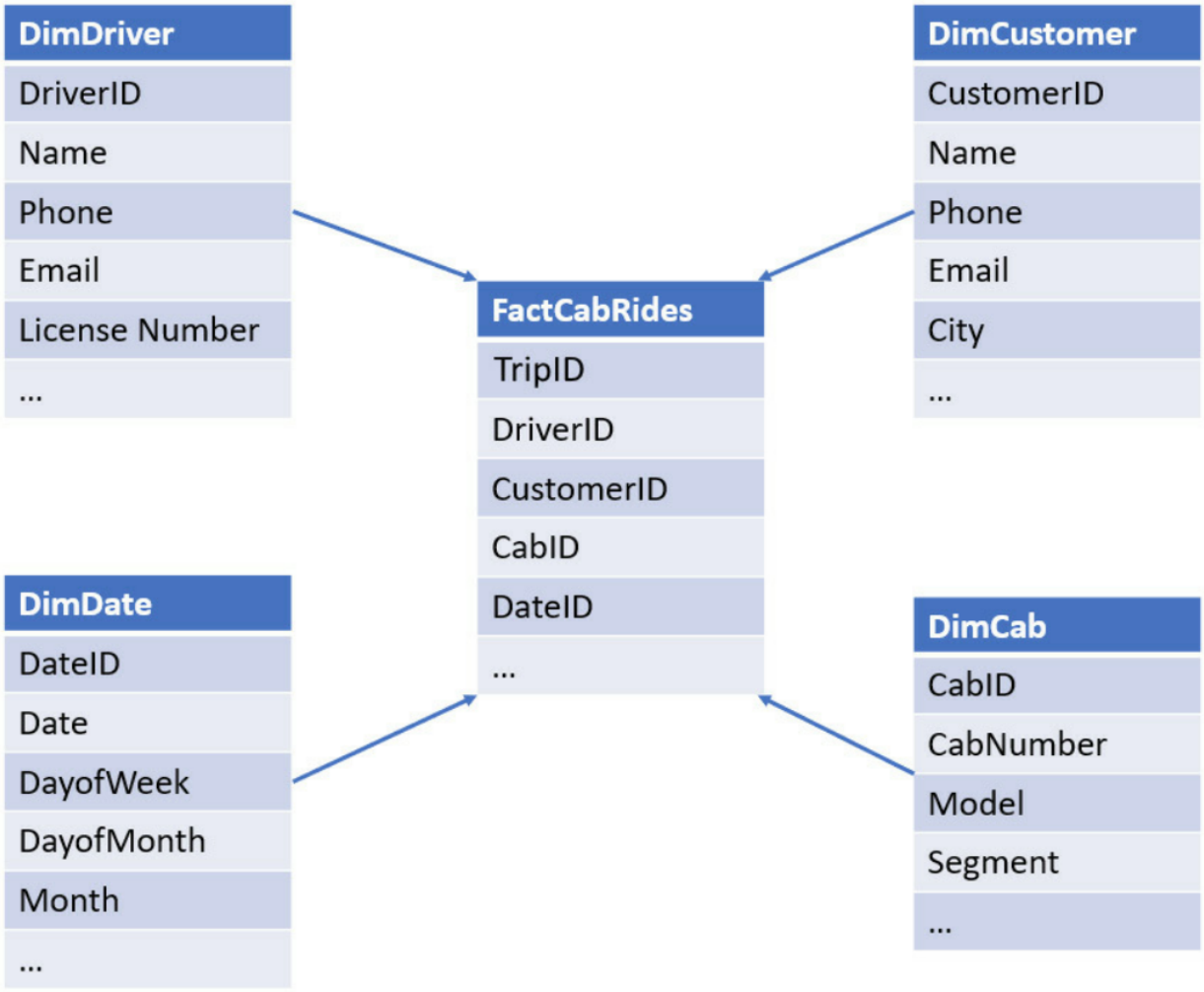


Figure 4.1 – Example of a Star schema

Snowflake Schemas

- Fact tables remain the same but dimension tables are further split into normalized forms
- Normalized dimension tables are referenced with foreign keys
- Normalized dimension tables minimize duplication across tables
- Best used when there are BI and non-BI applications sharing the same warehouse
- Important points about snowflake schemas
    - Fact tables usually have much higher volume
    - Dimension data is normalized, avoiding redundant data
    - Dimension tables can be connected to each other via foreign keys
    - Data optimized for storage and integrity, not speed
    - More complex than star schema, not the preferred option for BI use cases

    - Queries are usually slower due to multi-level joins
- In the below example, here are a few key relationships
    - Tables: DimLicense > DimDriver > FactCabRides
        - Foreign Key: LicenseID
        - Primary Key: DriverID

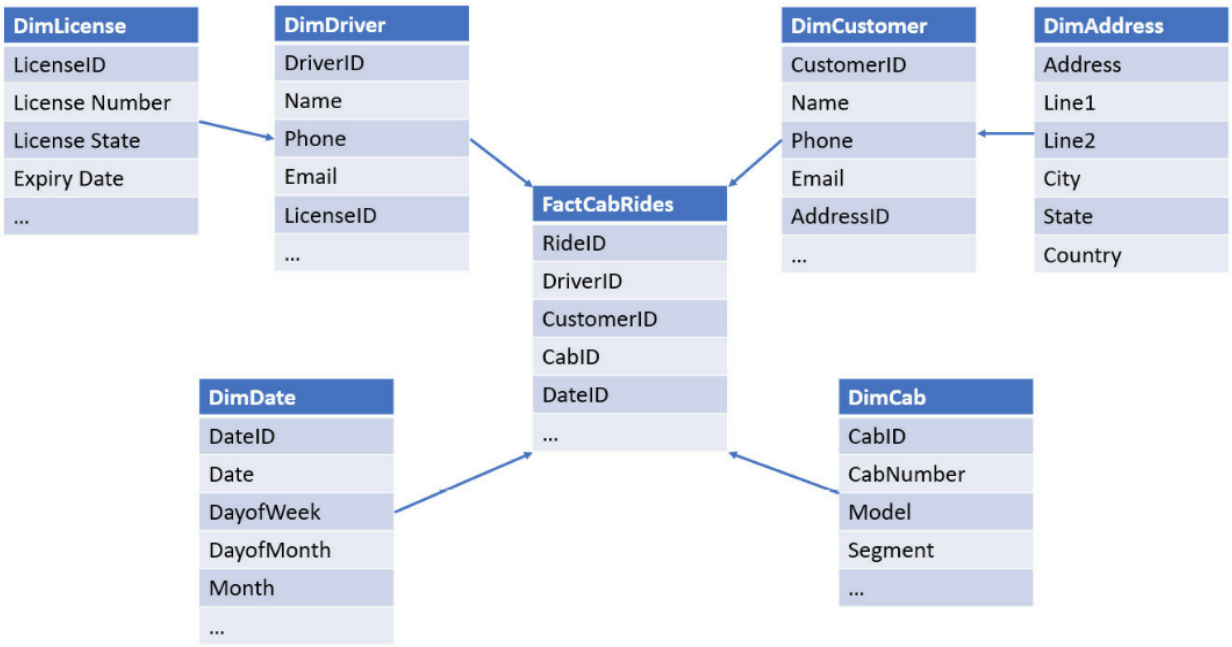


Figure 4.2 – Example of a Snowflake schema

Designing Slow Changing Dimensions (SCDs) (Pg 82)

- SCDs refers to dimension data that changes slowly over time
    - Example, customer email address or phone number
- Things to consider when designing SCDs
    - Do changes need to be tracked or discarded
    - How much history should be maintained or ignored
- Roughly seven types: SCD1 through SCD7
- SCD1, SCD2, SCD3, SCD4, and SCD6 are most common

Designing SCD1

- Values are overwritten and no history is maintained
- Once data is updated, there is no way to find out what the previous value was

Designing SCD2

- Complete history of changes are maintained
- A new row is added with all attributes whenever there is a change
- Common approaches to identifying which row is the most recent record
    - isActive flag (boolean)
        - Filter: isActive = 1
    - Version flag (integer)
        - Filter: MAX (Version)
    - StartDate and EndDate flag
        - Filter: EndDate IS NULL
- Surrogate key is added to all records as a secondary row identification key
    - Primary identification key is no longer unique in SCD2 design

Designing SCD3

- Partial history maintained
- Instead of adding additional rows, an extra column is added with the previous value
- Surrogate keys no longer needed, only one record per dimension is maintained so primary key is unique
- Example attributes: City and PrevCity
    - Only two most recent cities are maintained

Designing SCD4

- Introduced for dimension attributes that change frequently
- Fact-changing attributes of dimension table are split into new dimension tables
- Dimension tables reference the fact tables via primary keys (star schema)

Designing SCD6

- Combination of SCD1, SCD2, and SCD3
- Along with addition of new rows, the latest value is updated in all rows
- Example attributes: CurrCity, PrevCity, StartDate, EndDate, IsActive

Surrogate ID	Customer ID	Name	CurrCity	PrevCity	StartDate	EndDate	isActive
1	1	Adam	Miami	NULL	01-Jan-2020	25-Mar-2020	False
2	1	Adam	Miami	New York	25-Mar-2020	01-Dec-2020	False
3	1	Adam	Miami	New Jersey	01-Dec-2020	NULL	True

Figure 4.11 – Example of SCD type 6

Designing a Solution for Temporal Data

- Temporal data refers to data at a specific point in time
- Temporal tables are specialized tables that keep track of data changes over time
- Storing temporal data is required in situations like
  - Data auditing
  - Forensic investigations
  - Maintaining SCDs
  - Point in time recoveries, etc
- Available in Azure SQL Database and SQL Server (not available in Synapse pools as of the writing of this book)
- Behind the scenes, two tables are created
  - Temporal table (the one defined)
  - History table
  - Current values are stored in the temporal table
  - Old values get moved to the history table with the end time updated to the current time stamp (indicating row is no longer active)
- Example query of a temporal table showing history

```
33  SELECT [customerId]
34      , [name]
35      , [address]
36      , [validFrom]
37      , [validTo]
38      , IIF (YEAR(validTo) = 9999, 1, 0) AS IsActual
39  FROM [dbo].[Customer]
40  FOR SYSTEM_TIME BETWEEN '2022-01-22' AND '2022-01-24'
41  WHERE CustomerId = 101
42  ORDER BY validFrom DESC;
43
```

Results Messages

Search to filter items...

customerId	name	address	validFrom	validTo	IsActual
101	Alan Li	111 Updated Lane, LA	2022-01-23T07:38:32....	9999-12-31T23:59:59....	1
101	Alan Li	101 Test Lane, LA	2022-01-23T07:36:26....	2022-01-23T07:38:32....	0

Figure 4.12 – Sample result of a Temporal query

Designing a Dimensional Hierarchy

- Dimensional hierarchy refers to the way we group and organize dimensional data at multiple levels
- Main characteristics of hierarchical structure is that all nodes are identical, and they include pointers to their parent or children nodes
- Example hierarchical structures
    - Organizational structures
    - Product categories
    - File systems
- Dimensional hierarchy is achieved by using self-joins within a dimension table (self-referencing relationship)
- In the below example, a parent key is added which references the employees manager
    - Parent keys should allow NULLs as the root element of the hierarchy will not have a parent (such as the CEO of a company)

```
5   CREATE TABLE DimEmployee (
6       [employeeId] VARCHAR(20) NOT NULL,
7       [name] VARCHAR(100),
8       [department] VARCHAR(50),
9       [title] VARCHAR(50),
10      [parentEmployeeId] VARCHAR(20)
11  )
12
```

Results Messages

Search to filter items...

employeeId	name	department	title	parentEmployeeId
100	Alan Li	Manufacturing	Manager	
200	Brenda Jackman	Manufacturing	Supervisor	100
300	David Hood	Manufacturing	Machine operator	200

Figure 4.13 – Sample Hierarchical dimension table

Tip: If you are using Dimensional Hierarchy as part of an SCD, always add the Parent key pointing to the Surrogate Key instead of the Business primary key. Because surrogate keys will be unique, and it will ensure that the dimensional hierarchy doesn't break when changes to the business key happens.

Designing for Incremental Loading

- Incremental loading or delta loading refers to the process of loading smaller increments of data into a storage solution
- During data ingestion scenarios, it is very common to do a bulk upload followed by scheduled incremental loads
- Azure Data Factory (ADF) can help with designing incremental loads
    - ADF is a managed cloud service used to coordinate and orchestrate complex pipelines
    - ADF is built on the following components
        - Pipelines: A collection of activities linked together to perform some control flow or data transformation
        - Activities: Steps in the pipeline such as copying data, running a Spark job, etc
        - Datasets: Data that pipelines and activities operate on

        - Linked Services: Connections to data stores and computes in Azure (like connections strings that let you access external sources)
        - Triggers: Events used to start pipelines or activities
- Based on the data source, we can implement the following incremental loading techniques
    - Watermarks: Relational data source
    - File Timestamps: Blob data source
    - Partition Data: Time partitioned data source
    - Folder Structure: Time divided data source

Watermarks (Pg 95 for ADF Example)

- Watermarking tracks the last record loaded (our watermark) and loads all new records beyond the watermark on the next incremental run
- In relational stores, watermark details can be stored in a table and updated with stored procedures
    - Every time a new record is loaded, the stored procedure would trigger and update the watermark table
    - The next load can use this watermark information to identify the new records that need to be loaded

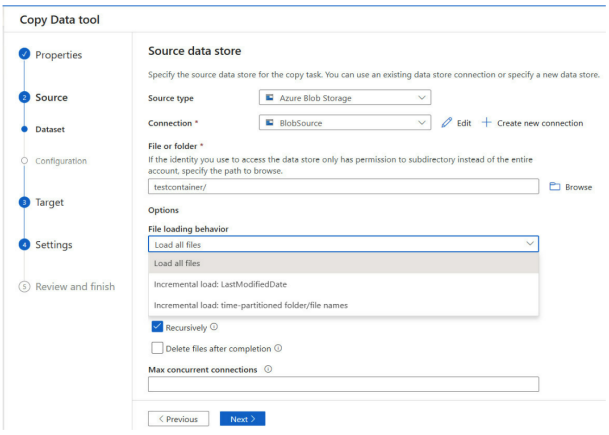
File Timestamps (Pg 99 for ADF Example)

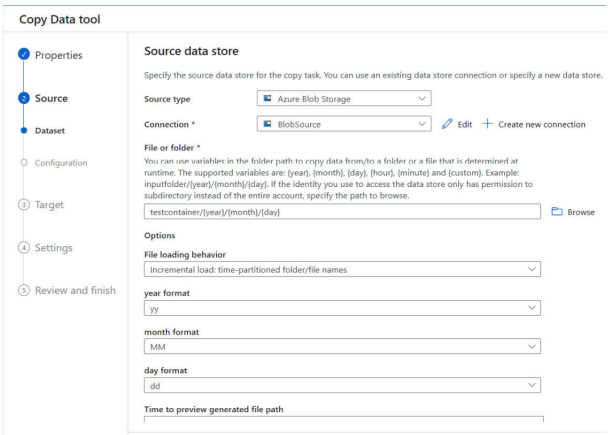
- ADF's Copy Data tool provides an option to scan source files based on the LastModifiedDate attribute
- Once the source and destination folders are specified and Tumbling Window is selected, we can select Incremental Load: LastModifiedDate

File Partitions and Folder Structures (Pg 101 for ADF Example)

- ADF's Copy Data tool can also be used for file partitioning and date-based folder structure use cases
- Once the source and destination folders are specified and Tumbling Window is selected, we can select Incremental Load: Time-Partitioned Folder/File Names

ADF Copy Data Tool for Incremental Loading





Designing Analytical Stores

- Analytical stores can be SQL or NoSQL stores deployed in the data lake Serving Zone
- Main job of an analytical data store is to serve data to BI tools
- Azure storage options that can cater to these requirements
    - Azure Synapse Analytics
        - Serverless SQL Pools: Ad-hoc analysis
        - Spark Pools: Support analytical workloads
    - Azure Databricks
    - Azure Cosmos DB
    - Azure SQL Database
    - HBase/Phoenix and Hive LLAP on HDInsight
    - Azure Data Explorer
- Reference Pg 104-105 for security and scalability considerations

Designing Metastores in Azure Synapse Analytics and Azure Databricks

- Metastores store the metadata of data in services such as Spark or Hive
- Metastores are similar to data catalogs which tell you
    - Which tables you have
    - What the table schemas are
    - What the relationships between the tables are
    - Where the tables are stored, etc

- Spark supports two metastore optioins
  - In-memory metastores
  - External metastores

Summary

That brings a close to our fourth chapter. Congratulations on making it this far.

Just to recap, we started off with the basics of data modeling and learned about Star and Snowflake schemas. We then learned about designing for SCDs, the different sub-types of SCDs, dimensional hierarchies, handling temporal data by using time dimensions, loading data incrementally using ADF, and selecting the right analytical store based on the customer's requirements. Finally, we learned about creating metastores for Synapse and Azure Databricks. All these topics complete the syllabus for DP203 – Design the Serving Layer. You have now learned how to design your own Serving layer in Azure.

We have now come to the end of our design chapters. We will be focusing on the implementation details from the next chapter onward. Yay!

Chapter 5: Implementing Physical Data Storage Structures

Implementing Compression

- As data lake size grows, it's important we store data in compressed format to save on cost
- We'll look at compressing data with ASA and Spark
- Azure Synapse Pipelines within ASA is the same as Azure Data Factory

Compressing Files using Synapse Pipelines or ADF

- Azure Blob Storage (Storage Account) and ADLS Gen 2 support the option of compressing

- In the Copy Data tool, select from the following compression types
  - bzip2
  - gzip
  - deflate
  - ZipDeflate, etc

Compressing Files Using Spark

- Spark provides libraries that can directly write the outputs in compressed formats such as Parquet, ORC, etc
- We can specify the compression algorithm to use in the script
- Spark supports the following compression types
  - gzip
  - snappy
  - lzo
  - lz4
  - brotli
  - zstd

Implementing Partitioning

- For storage based partitioning, the main technique is to partition the data into an optimal folder structure
- Azure recommends the following pattern
  - Region / Subject Matter(s) / YYYY / MM / DD / HH

Using ADF or Synapse Pipelines to Create Data Partitions

- Folder path settings can be set in the data flow activity under the sink options
- The Folder Path field accepts dynamic expressions
- The example below would create a folder path based on the current date, such as: staging / driver / out / 2023 / 01 / 01

```
"staging/driver/out/" + toString(year(currentDate()))  + "/" +
toString(month(currentDate())) + "/" + toString(dayOfMonth(currentDate()))
```

Partitioning for Analytical Workloads

- Three types of partitions for analytical workloads
  - Horizontal partitioning (sharding)
  - Vertical partitioning
  - Functional partitioning
- Horizontal partitioning techniques are dynamic and done after schemas are created
- Vertical or functional partitioning are done when the schema is built out
  - Implementation involves creating multiple tables and linking them using constraints (such as a foreign key constraint)
  - Foreign key constraint not yet supported in ASA SQL Pools

Implementing Horizontal Partitioning (Sharding)

Sharding in Synapse Dedicated Pools

- Synapse SQL Dedicated Pools support three types of tables
  - Clustered columnstore
  - Clustered index
  - Heap
- These tables three different sharding distributions
  - Hash
  - Round-robin
  - Replicated
- Important Note: Dedicated SQL Pools partition data into 60 partitions by default. Partitions specified explicitly will add to this, so be careful not to over-partition.

Using Dedicated SQL Pools to Create Sharding

- Dedicated SQL Pools have an option called DISTRIBUTION and PARTITION in the CREATE TABLE statement
- Specify a sharding distribution option in the DISTRIBUTION call (including a column for HASH)
- Specify a columns and RANGE RIGHT FOR VALUES or RANGE LEFT FOR VALUES (with values) in the PARTITION call
- Range right and left ensures the value specified in the PARTITION call will belong to the right or left side of the partition

Sharding Using Spark

- Spark by default partitions data based on the number of cores available (customizable)
  - Or the number of Hadoop Distributed File System (HDFS) blocks if running on HDFS
- If we need to partition the data in a custom format, Spark supports
  - In-memory partitioning
  - On-disk partitioning

In-Memory Partitioning

- Spark provides the following three methods to perform in-memory partitioning
  - repartition(n) increases the number of partitions
  - coalesce(n) decreases the number of partitions
  - repartitionByRange(n, col) repartition command with specified ranges

On-Disk Partitioning

- Spark provides the partitionBy(col1, col2...) operator
- Used to partition data and store it in different files while writing the output
- Folders are created for the column arguments

Implementing Distributions

- Dedicated SQL Pools (MPP) splits data into 60 partitions and executed them in parallel by default
- Each of these smaller partitions along with the compute resources used to run the queries are called a distribution
- A distribution is a basic unit of processing and storage for a Dedicated SQL Pool
- Dedicated SQP Pools support three distribution options
  - Hash
  - Round-robin
  - Replicated

Hash Distribution

- Distributes data based on a hash function

- Rows with the same value for the hashed column will always move to the same partition
- Specified with DISTRIBUTION = HASH (col) in the WITH clause of the CREATE TABLE statement
- Best used for tables greater than 2 GB or if the table is frequently updated

Round-Robin Distribution

- Distributes data randomly across all nodes in a round-robin fasion
- Specified with DISTRIBUTION = ROUND_ROBIN in the WITH clause of the CREATE TABLE statement
    - No column is specified
- Best used for data being loaded into staging tables or when there is no good indexing key available

Replicated Distribution

- Distributes a complete copy of the data across all nodes
- Specified with DISTRIBUTION = REPLICATE in the WITH clause of the CREATE TABLE statement
- Best used for small but frequently accessed tables

Implementing Different Table Geometries with Azure Synapse Analytics

- Geometries refers to features
- The main features of Synapse Dedicated Pool tables are
    - Partitions (covered above)
    - Distributions (covered above)
    - Indexes
- Dedicated SQL Pools support three types of indexing
    - Clustered columnstore indexing (default)
    - Heap indexing
    - Clustered indexing

Clustered Columnstore Indexing

- Default indexing options
- Column-based storage index that provides very high levels of compression and better query performance over row-based indexes
- Specified with CLUSTERED COLUMNSTORE INDEX in the WITH clause of the CREATE TABLE statement
- Best used for large fact tables (60M + records)

Heap Indexing

- Used as temporary data-loading tables as they provide faster loading times
- Specified with HEAP in the WITH clause of the CREATE TABLE statement
- Best used for staging data before loading into refined (curated, gold) tables
    - Works best with smaller tables

Clustered Indexing

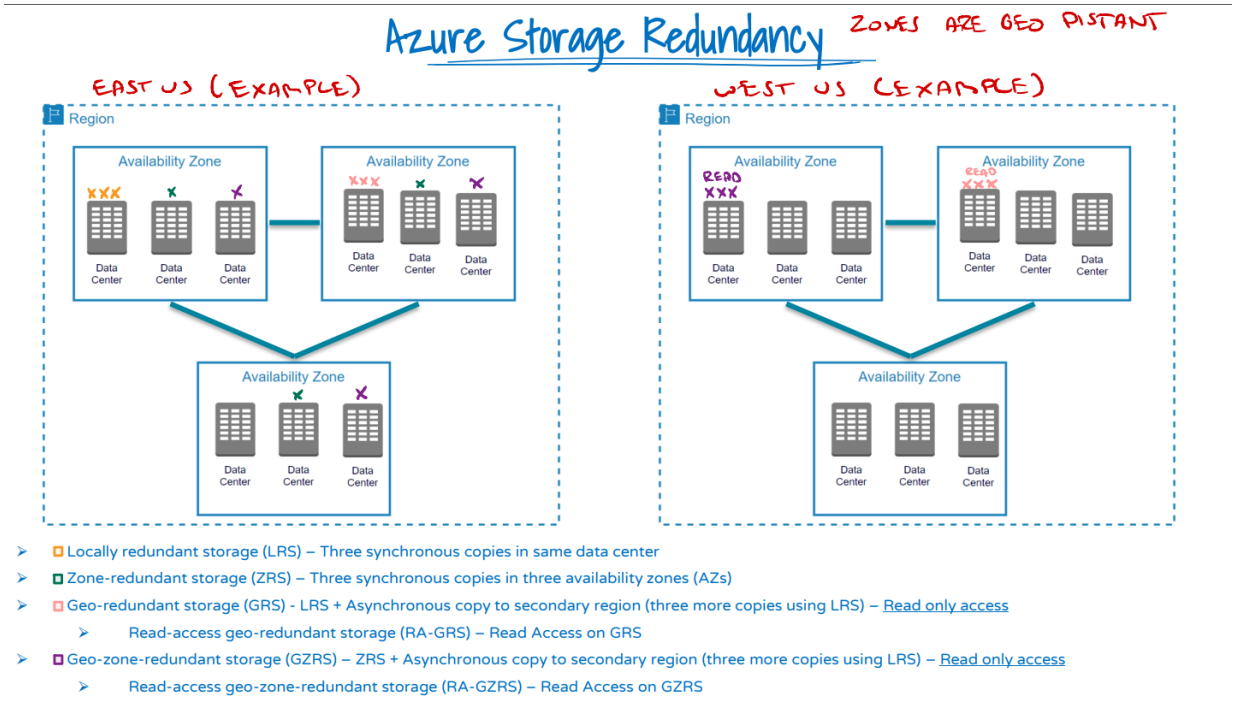
- Row-based storage index
- Faster for queries that need row lookups with highly selective filters on the clustered index column
- Specified with CLUSTERED INDEX (col) in the WITH clause of the CREATE TABLE statement

Implementing Data Redundancy

- Data redundancy is the process of storing multiple copies of data in different locations
- Protects data from events such as
    - Power failures
    - Disk failures
    - Network failures
    - Data center outages, etc
- Azure Storage provides two categories of data redundancy
    - Primary region redundancy
    - Secondary region redundancy
- Can be set when the storage account is created, or modified after creation
    - Changing redundancy after creation will require manual or live migration
    - Meaning data will have to be copied manually or automated with ADF

- Not all Azure services or regions support all four redundancies (LRS, ZRS, GRZ, GZRS)

Azure Storage Redundancy



- Locally redundant storage (LRS) – Three synchronous copies in same data center
- Zone-redundant storage (ZRS) – Three synchronous copies in three availability zones (AZs)
- Geo-redundant storage (GRS) - LRS + Asynchronous copy to secondary region (three more copies using LRS) – Read only access
  - Read-access geo-redundant storage (RA-GRS) – Read Access on GRS
- Geo-zone-redundant storage (GZRS) – ZRS + Asynchronous copy to secondary region (three more copies using LRS) – Read only access
  - Read-access geo-zone-redundant storage (RA-GZRS) – Read Access on GZRS

Azure Storage Redundancy in the Primary Region

- Covers localized failures such as
  - Disk failures
  - Machine failures
  - Rack failures
- Two types
  - Locally Redundant Storage (LRS)
  - Zone Redundant Storage (ZRS)

Locally Redundant Storage (LRS)

- Data is copied three times across multiple machines in the same physical region with a data center
- If all three data copies end up in the same rack, data loss can occur if the rack goes down
- Cheapest of all redundant-storage options

Zone Redundant Storage (ZRS)

- Data is copied across multiple Availability Zones (AZs)
- AZs are local regions within a data center that have different power and networking supplies
- If one zone goes down, the other can continue serving data
- More expensive option than LRS

Azure Storage Redundancy in Secondary Regions (Covered Above)

- Provides data protection against major geographical failures
- Covers region failures such as
  - Total data center outages caused by massive power outages
  - Natural catastrophes, etc
- Data is copied locally, and in data center at a different geological location
- If primary region goes down, data can automatically or manually fail over to the secondary region
- Two types
  - Geo Redundant Storage (GRS)
  - Geo-Zone Redundant Storage (GZRS)

Geo Redundant Storage (GRS)

- This option is equivalent to an LRS at a primary and secondary region
- Data is first written in LRS fashion at the primary region, and then a remote copy is made asynchronously at the secondary region
- At the remote site (secondary region), data is again copied three times using local LRS
- Azure provides a read-only extension called Read-Access GRS (RA-GRS)
  - Only difference is the remote copy is read-only

Geo-Zone Redundant Storage (GZRS)

- This option is equivalent to ZRS at the primary region, and LRS at the secondary region
- Local copies are synchronously copied at a primary region
- Remote copies are asynchronously copies at a secondary region
- The secondary region keeps a LRS copy, but not a ZRS copy

- Azure provides a read-only extension called Read-Access GZRS (RA-GZRS)
  - Only difference is the remote copy is read-only
- Highest data redundancy option, and the most expensive

Azure SQL Geo-Replication

- Feature called Active Geo Replication
- Replicates the complete Azure SQL instance to the secondary servers configured in a different geographical region
- Secondary servers act as read-only servers
  - Can be switched to read-write during failovers

Azure Synapse SQL Data Replication

- Synapse does geo-backups to its paired data centers every 24 hours by default (can opt out)
- We can restore from a geo-backup to any Synapse SQL pool, in any region

Cosmos DB Data Replication

- CosmosDB is a globally distributed database (inherently redundant)
  - Transparently replicates data to configured geographical locations
- CosmosDB also automatically backs up data periodically
  - These backups can be used to recover in case of accidental deletes or account upgrades

Implementing Data Archiving

- Azure Storage provides options to create data lifecycle policies that move data into different tiers based on age
- Three tiers available
  - Hot
  - Cold
  - Archive

Summary

That brings us to the end of this chapter. I hope you enjoyed it as much as I enjoyed writing it. We started with learning how to compress data in a clean way using Synaspe Pipelines and ADF, and natively using Spark. Then, we focused on learning about implementing partitioning, sharding, and distributions in SQL dedicated pools and Spark. After that, we learned about the different types of tables and indexing available in SQL dedicated pools, and finally, we concluded with learning how to set up data redundancy and archiving.

All the preceding topics should cover the syllabus of DP203 – Implement Physical Data Storage Structures. We will focus on implementing logical data storage structures in the next chapter.

Chapter 6: Implementing Logical Data Storage Structures

- Implement advanced data loading concepts
  - Slowly changing dimensions
  - Storage based solutions for optimizing query performance
  - Techniques to read external data without copying/loading to local storage

Building a Temporal Data Solution

- Concept is reviewed in Chapter 4, Section: Designing a Solution for Temporal Data
- Temporal data refers to data at specific points in time
- Temporal solutions deal with building systems that can handle and store time based data

Building a Slowly Changing Dimension

- Concept is reviewed in Chapter 4, Section: Designing a Solution for Slowly Changing Dimensions
- SCDs refers to dimension data that changes slowly over time
  - Example, customer email address or phone number
- Things to consider when designing SCDs

    - Do changes need to be tracked or discarded
    - How much history should be maintained or ignored

Implementation Overview (Pg 136)

- Implementation can be done in ADF or Synapse Pipelines (among others)
- The book reviews building an SCD2 solution in ADF/Synapse Pipelines (Synapse Pipelines is just an implementation of ADF)
    - Logic implemented using three data flows within a pipeline
        - Logic for new rows
        - Logic for modified rows
            - Update latest row as active
            - Update previous rows as inactive
- Reference book for implementations teps

Designing SCD2 (Refresher)

- Complete history of changes are maintained
- A new row is added with all attributes whenever there is a change
- Common approaches to identifying which row is the most recent record
    - isActive flag (boolean)
        - Filter: isActive = 1
- Surrogate key is added to all records as a secondary row identification key
    - Primary identification key is no longer unique in SCD2 design

Building a Logical Folder Structure

- Concept is reviewed in Chapter 2, Section: Designing Folder Structures for Data Transformations
- Rule of thumb is to store data in a hierarchical date folder structure (date at end)
- Folder structure recommended by Microsoft
    - Region / Subject Matter(s) / YYYY / MM / DD / HH
- Folder structure recommended by Microsoft for sensitive data use cases
    - Region / General / Subject Matter(s) / YYYY / MM / DD / HH
    - Region / Sensitive / Subject Matter(s) / YYYY / MM / DD / HH

Implementing File and Folder Structures for Efficient Querying and Data Pruning

- Concept is reviewed in Chapter 2, Section: Designing Storage for Data Pruning
- Once we have a date-based hierarchical folder structure, query performance can be improved via data partitioning
- If data is divided into partitions stored in different folder structures, queries can skip scanning irrelevant partitions (considered data pruning)
- Additionally, partition increases efficiency of data loading and deletion by performing partition switching and partition deletion
- Partitions have to be aligned perfectly on the boundaries for partition switching

Deleting an Old Partition (Pg 149)

- To delete a partition, we need to create a dummy table that has the same structure as the original table
- We swap out the partition we want to delete to the dummy table
- The original table will now contain 0 records for that partition

Adding a New Partition (Pg 149)

- To add a new partition, we need to split the last partition into two partitions using the SPLIT clause in the ALTER TABLE command
- Before we split, we need to move the data in the last partition to a dummy table as we can't split a partition that contains data
- SPLIT will split the last partition into the previous and new partition
    - Example: Split Partition 3 into Partition 3 and 4
- We then move the partitions back to the source table using the SWITCH PARTITION clause in the ALTER TABLE command
- ALTER TABLE commands execute almost immediately as they are metadata operations

Building External Tables (Pg 151)

- External tables are similar to regular tables except they are stored in external storage locations
- External tables mitigate having to copy data into internal tables for processing

    - They can directly read data from external sources, saving on data transfer cost
- Synapse Dedicated SQL Pools and Serverless SQL support external tables
- The following three constructs have to be defined in order to access data via external tables
    - EXTERNAL DATA SOURCE
    - EXTERNAL FILE FORMAT
    - EXTERNAL TABLE
- External tables can also be created in the Synapse GUI (once options are selected, Synapse will create the required script)

Summary

That brings us to the end of this chapter. This was a smaller chapter but it had some very important concepts from the certification's perspective. We started by looking at building SCDs, which was a lengthy but relatively easy process as we mostly had to just drag and drop the components into Synapse pipelines and configure them. Then, we revisited the general rule of thumb for building an efficient folder structure. We also learned how to quickly switch partitions in and out using Synapse SQL. Finally, we learned how easy it is to create external tables in Synapse.

All the topics that were covered in this chapter should cover the syllabus for DP203 – Implementing the logical data structures. In the next chapter, we will focus on implementing the serving layer.

Chapter 7: Implementing the Serving Layer

- Implementing a serving layer involves
    - Implementing star schemas
    - Techniques to read different data formats
    - Sharing data between services such as SQL and Spark

Delivering Data in a Relational Star Schema

- Star schemas have two types of tables
    - Fact tables

    - Dimension tables
- Fact tables are much larger and benefit from hash distribution with columnstore indexing
- Dimension tables are smaller and change less frequently, benefiting from replicated table distribution
- Implementing fact and dimension tables (Pg 136)
    - CREATE TABE statements with the appropriate distribution and indexing options
    - Apply primary keys to link tables, example below (column names spaced for readability)
        - Fact Trips Table
            - Trip ID
            - Driver ID
            - Customer ID (PK to Dim Customer)
            - Trip Date (PK to Dim Date)
            - Start Location
            - End Location
        - Dim Customers Table
            - Customer ID (PK to Fact Trips)
            - Name
            - Email ID
        - Dim Date Table
            - Date ID
            - Date (PK to Dim Date)
            - Day Of Week
            - Fiscal Quarter

Implementing a Dimensional Hierarchy

- Concept is reviewed in Chapter 4, Section: Designing a Dimensional Hierarchy

Delivering Data in Parquet Files

- Learn to deliver data present in Parquet files using Synapse SQL and Spark
- Uses the concept of external tables, we can create and query data from Parquet files easily

- Using Synapse SQL Serverless (Pg 160)
- Using Synapse Spark (Pg 162)
- Using Azure Databricks Spark (Pg 162)
    - Similar to Synapse Spark
    - Will need to define a service principal for Azure Databricks to connect with Azure Data Lake Storage Gen2
- All three options support charting query results for ad-hoc visualizations of data

Maintaining Metadata

- Concept is reviewed in Chapter 4, Section: Designing Metastores
- Metastores are like catalogs that contain information about all
    - Tables
    - Table schemas
    - Table relationships
    - Storage location

Metadata Using Synapse SQL Serverless and Spark Pools (Pg 164)

- Synapse supports a shared metadata model
- Databases and tables that use Parquet or CSV storage formats are automatically shared between compute pools (SQL Built-In Serverless and Spark)
    - Data created from Spark can only be read and queried by SQL pool, but cannot be modified at the time of this book
- Shared data model makes it easy to share and query data between SQL and Spark Pools, example below
    - Create a database and table using Synapse Spark Pool (Parquet or CSV format specified in the USING clause)
    - Create a SQL script and connect to Built-In pool
    - Query Built-In pool using FROM SparkDatabase.Schema.Table

Metadata Using Azure Databricks (Pg 167)

- In order to share data between Spark and other services outside of Synapse, we have to use the Hive metastore
    - Using Databricks as the Spark engine is outside of the Synapse workspace

- Example sharing data between Azure Databricks Spark and Azure HDInsight Hive (Pg 167)
    - Create a SQL Database resource
    - Create a HDInsight resource and choose Hadoop or Interactive Query for cluster type (both will install Hive)
        - Select the SQL Database created above in the External
    - View SQL Database contents in the HDInsight Ambari dashboard (Views: Hive 2.0)
    - Create Azure Databricks cluster using config settings from book
        - JDBC connection string from SQL Database will be needed
    - View SQL Database contents directly from a Databricks Spark notebook

Summary

This was another small but interesting chapter. We started with implementing the star schema, then learned about delivering data in Parquet format, and, finally, looked at how we can share data between the SQL-Spark and Spark-Hive services in Azure. With all this knowledge, you should now be able to design and implement a basic serving layer for a data lake architecture using Azure services. To learn more, please follow the links that I have provided at the end of important topics.

With this, we have reached the end of the first major section of the DP-203 syllabus, Designing and Implementing Data Storage. This covers about 40–45% of the certification examination. We are getting closer to the halfway mark. Good going!

In the next section, we will learn about designing and developing data processing systems and, more specifically, about ingesting and transforming data in data lakes.

Chapter 8: Ingesting and Transforming Data

- Data transformation is the process of transforming data from its raw format to a more useful format
- Transformed data can be used by downstream tools for business intelligence, data analysis, and data science
- This chapter covers

    - Reading different file formats and encodings
    - Data cleansing
    - Transformations
    - Services: Spark, SQL, ADF, Databricks

Transforming Data Using Apache Spark

- Apache Spark supports transformations with three different Application Programming Interfaces (APIs)
    - Resilient Distributed Datasets (RDDs)
    - DataFrames
    - Datasets

What Are RDDs

- Fundamental data structure used by Spark
- RDDs are immutable, fault-tolerant, collections of data objects
- RDDs can be operated on in parallel using Spark
- RDDs support a wide variety of data formats
    - JSON
    - CSV
    - Parquet, etc
- RDDs can be created using the parallelize() function
- RDD common transformation functions
    - map() Applies the function provided as a parameter to all elements in the source and returns a new RDD
    - flatMap() Similar to map(), but can apply the function to more than one element
    - filter() Returns a new RDD that satisfies the filter condition
    - groupByKey() Collects identical data into groups and can perform aggregate actions to the groups
    - union() Returns a new RDD that is a union of two datasets
    - distinct() Returns a new RDD that contains only the unique elements from the input

What Are DataFrames

- DataFrames are similar to tables in a relational database
- DataFrames are immutable, redundant, and distributed
- DataFrames contain schemas, columns, and rows
- Useful for processing large volumes of data while using relational table-like operations
- DataFrames can be created using the createDataFrame() and toDF() function
    - Can also create DF from CSV or JSON using spark.read.csv() and spark.read.json() functions
- DataFrames can be created with a schema using the schema parameter
- DataFrame transformations are more relevant than RDD transformations as they deal with table-like abstractions
    - A DataFrame transformation is eventually converted into an RDD transformation within Spark
- DataFrame common transformation functions
    - select() Select specified columns from the input
    - filter() Filter rows based on a condition
    - distinct() Select unique rows from the input
    - orderBy() Sort rows by a particular column
    - join() Join two tables based on specified conditions
    - groupBy() Collects identical data into groups and can perform aggregate actions to the groups
    - avg() Aggregate function to average values

Transforming Data Using T-SQL

- T-SQL is a procedural language
- Common transformation statements
    - SELECT
    - ORDER BY
    - DISTINCT
    - GROUP BY
    - UNION
    - JOIN
    - CREATE VIEW

Transforming Data Using ADF or Synapse Pipelines

- All ADF transformations happen on datasets, so we have to create source and sink datasets
- We will also have a create linked service
- ADF provides a code-free transformation option called Mapping Data Flows, which consists of three transformation types
    - Schema transformations
    - Row transformation
    - Multi-input/output (IO) transformations

Schema Transformations

- Actions that result in changing the schema of a table of DataFrame
- Commonly used schema transformations
    - Aggregates: Min, Max, Sum, Count, etc
        - Aggregate transformations will only output the columns used in the aggregation
        - Perform a self-join with the source data after this change if you want to return other columns
    - Derived Column: Add new column(s) to existing data
    - Select: Select required columns from input, or change column names before storing in sink dataset

Row Transformations

- Transformations that apply to the rows of the table
- Commonly used row transformations
    - Alter Row: Insert, delete, update, and upsert rows
        - Alter row only works on databases, CosmosDB, or Representational State Transfer (REST) endpoint sinks
    - Filter: Filter rows based on a condition
    - Sort: Sort rows on a column or group of columns

I/O Transformations

- Transformations that operate on more than one input, or split the input into more than one output
- Common I/O transformations
  - Join: Join streams based on one or more conditions
    - Full outer join
    - Inner join
    - Left outer join
    - Right outer join
    - Cross join
  - Union: Merges two input datasets with the same schema into one

Transforming Data Using Stream Analytics

- Stream analytics setup, concepts, and transformations will be covered in Ch 10: Designing and Developing a Stream Processing Solution

Cleansing Data

- Cleaning data refers to correcting anomalies as data flows from various sources do not conform to existing schemas
  - Missing values
  - Non-standard entries
  - Duplicates, etc

Handling Missing/NULL Values

- We can handle missing or NULL values by
  - Substituting missing values with default values using the Derived Columns transformation
    - Can also substitute with a meaningful value such as mean, median, average, tc
  - Filtering out NULL values using the Alter Row transformation

Trimming Inputs

- Values with trailing whitespaces are a common problem
- We can trim values using the trim() method in the Derived Column transformation

Standardizing Values

- Different input sources might use different conventions for the data
- We should standardize varying inputs before sending downstream for further processing
- A common approach is using the replace() function in the Derived Column transformation

Handling Outliers

- If values look abnormal relative to other values in the same field, we can replace them with average or median values
    - Be careful applying this transformation step and make sure to document it
- We can use the Aggregate transformation or Derived Column transformation with an expression
    - Example Derived Column transformation expression
    - IIF({Salary} > 5000, toInteger({AvgSalary}), {Salary})
        - This replaces the value in the Salary column with the average salary whenever a value if greater than 5000
        - This can lead to a host of complications
        - It is better to understand if this is an error or a valid anomaly
            - Errors should be fixed at the source if possible

Removing Duplicates (Deduping)

- Duplicates can be removed using the first() function in the Aggregate transformation
- Aggregate transformation by default only returns the column being aggregated
    - We can perform a self join after this transformation to return other columns
    - Or we can add name != 'Column Name being Transformed' in the Each COlumn Matches option
        - name != 'DriverID'

Splitting Data

- Common ways to split data in ADF include the following transformations

    - Conditional Split: Splits data based on certain conditions
    - New Branch: Copies the entire dataset for a new execution flow
- ADF also provides an option to split input data into multiple sub-files using partitions
    - Create a new Sink artifact
    - Specify the number of partitions required and partition type in the Optimize tab

Shredding JSON

- Shredding refers to the process of extracting data from JSON files into tables
- Shredding can be performed using Spark, SQL, or ADF

Extracting Values From JSON Using Spark

- Spark can directly read JSON files and extract the schema using the spark.read.json() function
- Schemas can also be specified manually using the StructType() function with .add() functions following

Extracting Values From JSON Using SQL

- T-SQL provides the OPENROWSET function to query remote data sources
- Parameters include
    - BULK (File Path)
    - FORMAT
    - FIELDTERMINATOR
    - FIELDQUOTE
- To extract from a JSON file, the FORMAT must be set to 'csv'

```
SELECT
    JSON_VALUE(doc, '$.firstname') AS firstName,
    JSON_VALUE(doc, '$.lastname') AS lastName,
    CAST(JSON_VALUE(doc, '$.id') AS INT) as driverId,
    JSON_VALUE(doc, '$.salary') as salary
FROM OPENROWSET(
        BULK '<INSERT https:// JSON LOCATION>',
        FORMAT = 'csv',
```

```
        FIELDTERMINATOR ='0x0b',
        FIELDQUOTE = '0x0b'
    ) WITH (doc NVARCHAR(MAX)) AS ROWS
GO
```

Extracting Values From JSON Using ADF

- ADF provides the Flatten transformation
- Converts hierarchical data structures (such as JSON) into flat structures (table)

Encoding and Decoding Data

- Collation defines the encoding and sorting type
- Encoding and decoding using SQL
  - Collation can be set at the database or table level using the COLLATE function
- Encoding and decoding using Spark
  - Spark supports encode(), decode(), and hex() methods
  - encode() and decode() can be passed into hex() to convert data to hexadecimal format (compact and readable)
- Encoding and decoding using ADF
  - Encoding can be set in the Dataset artifact under the Connection tab
  - ADF also supports functions for encoding and decoding
    -

`base64ToBinary()`	Return the binary version for a Base64-encoded string.
`base64ToString()`	Return the string version for a Base64-encoded string.
`decodeBase64()`	Return the string version for a Base64-encoded string.
`dataUriToBinary()`	Return the binary version for a data URI.
`dataUriToString()`	Return the string version for a data URI.
`decodeDataUri()`	Return the binary version for a data URI.

Configure Error Handling for Transformations

- ADF provides four different flows from an activity
  - Success
  - Failure
  - Completion
  - Skipped

- An error handling branch can be setup to either fix errors or save them for future actions
    - These if failure flows can be set up as a complete pipeline or single activity
    - Error details can also be configured to be insert into a database
- ADF Sink also provides options to write error lines to an external data store
    - Error Row Handling option under the Settings tab int he Sink activity

Normalizing and Denormalizing Values

- ADF activities that can normalize or de-normalize values
    - Flatten: De-normalize data
    - Pivot: De-normalize data
    - Unpivot: Normalize data
- The Pivot activity takes unique row values and converts them into table columns
    - Group By key
    - Pivot Key
- The Unpivot activity is the reverse of Pivot and is used to normalize data
    - Ungroup By key
    - Unpivot key

Performing Exploratory Data Analysis (EDA)

- Data exploration is much easier from inside the Synapse Studio
- Easy one-click options to look into various data formats

Data Exploration Using Spark

- From within Synapse Studio, right click on a data file and select Load to DataFrame
- This will create a new Notebook that can be ran and modified to explore the data

Data Exploration Using SQL

- From within Synapse Studio, right click on a data file and select Select TOP 100 Rows
- This will create a new script that can be ran and modified to explore the data

Data Exploration Using ADF

- ADF provides a Data Preview tab that works when the Data Flow Debug setting is turned on
- A small Azure Databricks cluster called the Integration Runtime runs behind the scenes and fetches real data
- This allows us to explore the data as we build pipelines and fine-tune transformations

Summary

With that, we have come to the end of this interesting chapter. There were lots of examples and screenshots to help you understand the concepts. It might be overwhelming at times, but the easiest way to follow is to open a live Spark, SQL, or ADF session and try to execute the examples in parallel.

We covered a lot of details in this chapter, such as performing transformations in Spark, SQL, and ADF; data cleansing techniques; reading and parsing JSON data; encoding and decoding; error handling during transformations; normalizing and denormalizing datasets; and, finally, a bunch of data exploration techniques. This is one of the important chapters in the syllabus. You should now be able to comfortably build data pipelines with transformations involving Spark, SQL, and ADF. Hope you had fun reading this chapter. We will explore designing and developing a batch processing solution in the next chapter.

Chapter 9: Designing and Developing a Batch Processing Solution

Designing a Batch Processing Solution

- Batch typically deals with larger amounts of data and takes more time to process when compared to stream processing
- Batch processing solutions typically consists of five major components
    - Storage systems

- Transformation/batch processing systems
- Analytics data stores
- Orchestration systems
- BI reporting systems

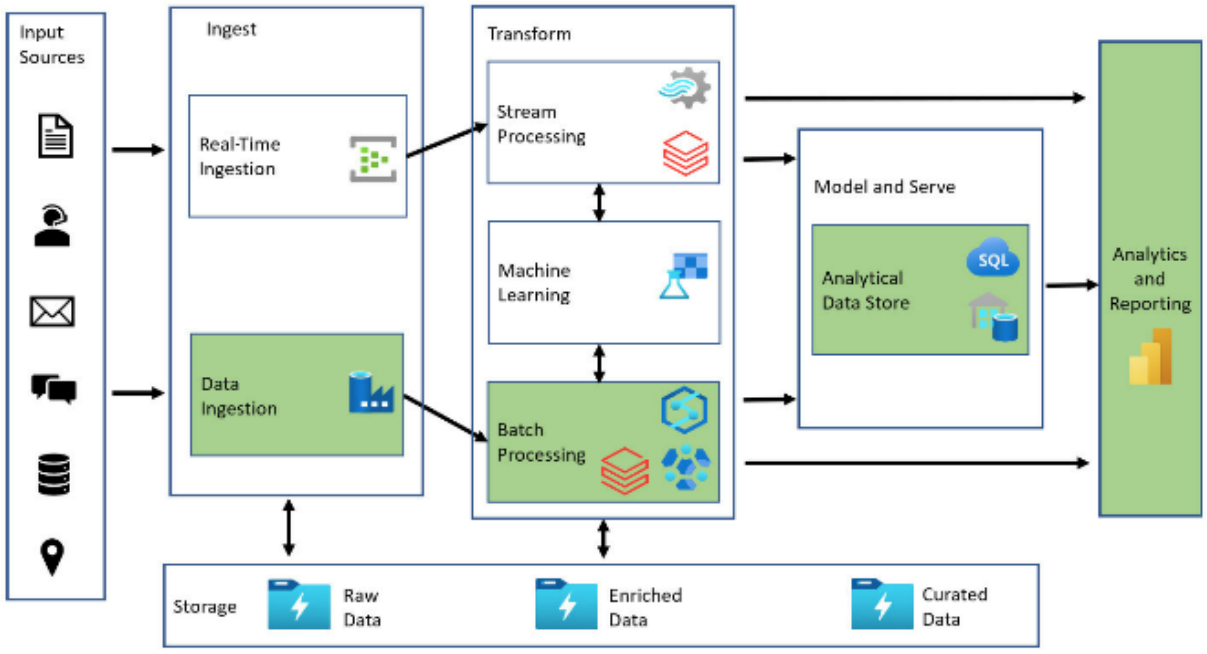


Figure 9.1 – Batch processing architecture

Developing Batch Processing Solutions

- Typical data pipelines will consist off five stages
  - Data ingestion
  - Data cleansing
  - Data transformation
  - Data loading (analytical database)
  - Data serving (BI tool)

Data Ingestion

- Process of landing raw data into an object store (ADLS in this example)
- Data from various sources will land in a data lake raw zone
- Some tool options available in Azure for ingestion
  - Azure Data Factory (ADF)
  - Azure Copy (AzCopy)

        - CLI tool suited for smaller data sizes (10-15 TB range)
    - Azure Express Route
        - Secure data transfers through dedicated private connections

Data Ingestion Example Using ADF (Pg 223)

- Create a Linked Service to connect to a blob store (ADLS)
- Create a data flow specifying data source and destination
- Create a data copy activity to copy data from source to destination

Data Preparation and Data Cleaning

- Reference Ch 8 Ingesting and Transforming Data
- Typical cleansing steps include
    - Deduping
    - Handling missing data
    - Handling outlier data (removal, replace, etc. Dependent on use case)

Data Transformation

- Transformation logic is ran once data is cleaned and prepped
- Several ways to write and run transformation logic such as
    - Spark
    - SQL
    - Hive, etc

Data Transformation Example Using Azure Databricks (Pg 227)

- Create a Azure Databricks (ADB) workspace and launch
- Create a cluster (required to run transformations in a pipeline)
- Create a notebook to write transformation logic
    - Write Spark code (supports Scala, Python, SQL, and R languages)
- Create a Linked Service to connect to a Databricks notebook
- Configure a ADB notebook activity in ADF (add Databricks activity and link to notebook with transformation logic)

Using PolyBase to Load Data into Analytics Data Store

- PolyBase enables services to copy and query data directly from external locations

- One of the fastest and most scalable ways to copy data
- Integrated into T-SQL
    - Automatically kicks in whenever we run COPY INTO < table > FROM < external storage location >
- Does not support nested data formats (JSON, XML, WinZip, etc)

Data Loading Example Using PolyBase (Pg 237)

- Copy transformed data from Azure Databricks into Synapse Dedicated SQL pool using a staging ADLS or blob store
    - Prepare source data in ADLS or blob store
    - Define external table in Dedicated SQL pool (make sure to create table with the correct schema)
    - Transform data into row and column format if source is non-relational
    - Run COPY INTO command to load data (PolyBase)
    - Copy data from external table to permanent SQL table
    - Serve data

Options for Loading with PolyBase

- PolyBase is also available as part of these services
    - Azure Data Factory
    - Azure Databricks
    - SQL Server (SSIS)

Data Serving

- There are many ways to serve data to end consumers
- Power BI is a MSFT data visualization tool that is deeply integrated with Azure
- PBI has built in connectivity to many cloud and on-premises technologies

Data Serving Example Using Power BI (Pg 239)

- Create a PBI workspace
- Create a Linked Service in Synapse for PBI
- In the Editor tab within the Synapse workspace
    - Click on the Power BI dropdown and create a new Power BI dataset
    - Download the file and open in PBI
- Work with the data in PBI to create a report or dashboard

- Alternatively, you can connect from within PBI by clicking Get Data
- Publish the report and grant access as needed

Creating Data Pipelines

- Data pipelines are a collection of data processing activities in a sequence
- Within Azure, we can create pipelines directly in Azure Data Factory or within the Synapse workspace (Synapse Pipeline)
  - Synapse Pipelines is an implementation of ADF within Synapse
  - There are many other technologies available to create, process, and orchestrate data pipelines
- Once a pipeline is created, it can be triggered one-time, event-based, or recurring on a schedule

Integrating Jupyter/Python Notebooks into a Data Pipeline (Pg 245)

- Jupyter notebooks can be integrated in an ADF pipeline using a Spark activity
- A linked service to Azure Storage and HDInsight is required, as well as a Spark cluster in HDInsight
- Utilizing a Jupyter notebook is similar to a Databricks notebook
  - We can write transformations in the Jupyter notebook and run using the ADF pipeline

Advanced Techniques for Data Loading and Preparation

- The following techniques are covered in other chapters
  - Designing and implementing incremental data loads (Ch 4)
  - Designing and developing slowly changing dimensions (Ch 4)
  - Handling duplicate data (Ch 8)
  - Handling missing data (Ch😎

Handling Late-Arriving Data in the Ingestion/Transformation Stage

- Drop data if application can handle data loss

- Rerun the pipeline from the Monitoring tab if the application cannot handle data loss

Handling Late-Arriving Data in the Serving Stage

- In the serving phase, data handling is usually done via a star or snowflake schema for OLAP scenarios
- There may be situations where a dimension or fact may arrive early or late
- Common methods for handling these situations
    - Drop data
    - Store data and retry at a later date
    - Insert a dummy record

Upserting Data

- Upserting refers to UPDATE or INSERT transactions
- ADF supports upsert operations if the sink is a SQL-based store
    - The sink activity must be preceded by an Alter Row operation (activity)
    - Allow Upsert option must be enabled in the sink activity
    - ADF will automatically do an upsert if a row already exists in the configured sink

Regressing to a Previous State

- Commonly used technique in databases and OLTP scenarios
- In OLTP scenarios, the entire transaction is rolled back if any transformation step fails (ACID)
- ADF does not have a ready-made option but we can create a rollback using the delete activity
    - The Copy activity has a Data Consistency Verification option in Settings
    - If the activity fails due to a consistency check, a follow up Delete activity can be created on the failure path (orange)
    - This will completely clean up the directory
    - The Recursively option in the delete activity will process all files in the input folder and its subfolders recursively

Introducing Azure Batch

- Azure Batch is a service that performs large-scale parallel batch processing
- Typically used for high-performance computing applications
- Consists of three main components;
    - Resource management
    - Process management
    - Resource and process monitoring
- Jobs can be run from the Azure portal or Azure CLI (Pg 253)
    - A job is a logical unit of work
    - Jobs are split into tasks and ran on parallel nodes

Configuring the Batch Size and Retention

- Batch size refers to both the size of Batch pools and the size of the VMs in those pools
- Points to consider when deciding on the batch size;
    - Application requirements
    - Data profile
    - Number of tasks that can be run per node (VM)
    - Different pools for different batch nodes
    - VM availability per region
    - VM quotas per Batch account
- Default batch retention in Azure Batch is 7 days unless the compute node is removed or lost
- The retention time can be changed while adding a job

Scaling Resources

- Scaling refers to the process of increasing or decreasing compute, storage, or network resources
- Aimed at improving job performance or reducing expenses
- Azure supports two types of scaling: Manual and Automatic
    - Manual scaling is set at resource creation and adjusted as needed by a user

    - Automatic scaling dynamically decides when to scale resources based on various factors
        - Cluster load
        - Cost of running cluster
        - Time constraints, etc.

Scaling in Azure Batch

- Azure Batch allows you to specify an auto-scaling formula based on the following
    - Time metrics
    - Resource metrics
    - Task metrics

Scaling in Azure Databricks

- While creating a cluster, Enable Autoscaling can be selected with Min and Max Workers specified
- The cluster will automatically scale between the Min and Max based on the load
- You can set a fixed-sized cluster by unchecking Enable Autoscaling and specifying the exact worker count
- An additional option is Spot Instances
    - These are unused Azure VMs offered at a lower cost with no availability guarantees
    - Azure can pull these resources with 30sec notice if needed
    - Only use this option for jobs that can handle interruptions such as large batch or dev/test jobs

Synapse Spark

- Similar to Azure Databricks
- Enable Autoscale and select a range for Number of Nodes

Synapse SQL

- Synapse SQL does not support autoscaling
- Instead an option is provided to choose the cluster performance level
    - The higher the number, the better the performance

Designing and Configuring Exception Handling

- Azure Batch provides error codes, logs, and monitoring events to handle and identify errors
- There are four common groups of errors;
  - Application errors
  - Task errors
  - Job errors
  - Output file errors
- Remedial action for exception handling is unique but the following are generic actions
  - Reboot the node
  - Reimage the node
  - Remove the node from the pool
  - Disable scheduling jobs on that pool

Best Security Practices for Azure Batch

- Use private endpoints
- Create pools in virtual networks
- Create pools without public IP addresses
- Limit remote access
- Encrypt data in transit
- Encrypt batch data at rest
- Apply compliance and security policies explained below

Handling Security and Compliance Requirements

- Azure provides a service called Azure Policy to enable and enforce compliance and security policies in any of the Azure services
- Azure Policy helps enforce policies and remedial actions at scale
- Pre-defined policy rules called Built-Ins are available for configuration
  - You can also create new policies

Azure Security Benchmark

- Azure provided recommendations to improve the security of cloud-based data, services and workloads
- You can validate services against these benchmarks to compare with industry standards

Summary

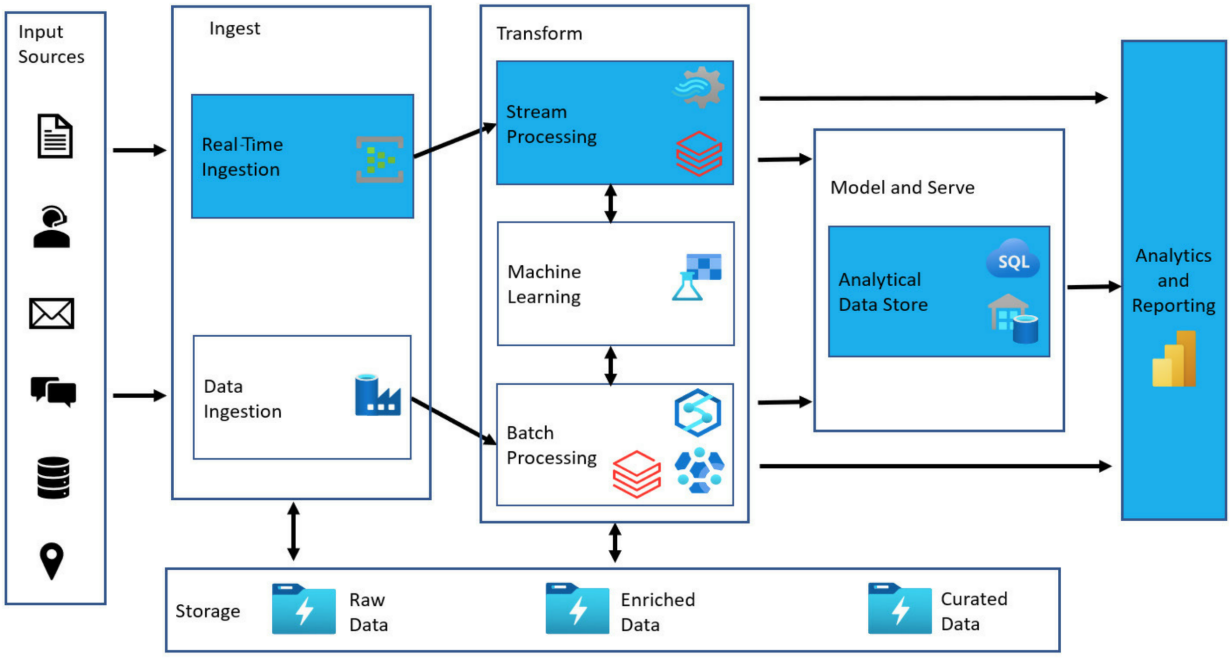
I hope you now have a good idea about both batch pipelines and the Azure Batch service. We learned about creating end-to-end batch pipelines by diving deep into each of the stages, such as ingestion, transformations, BI integrations, and so on. We then learned about a new service called Azure Batch and learned about batch retention, handling errors, handling autoscale, building data pipelines using Batch, and more. We also learned about some of the critical security and compliance aspects. That is a lot of information to chew on. Just try to glance through the chapter once again if you have any doubts.

We will next be focusing on how to design and develop a stream processing solution.

Chapter 10: Designing and Developing a Stream Processing Solution

Designing a Stream Processing Solution

- Incoming data is processed as it arrives with minimal latency
- Stream processing systems consists of four major components
    - Event ingestion service
    - Stream processing systems
    - Analytical data stores
    - Reporting systems



Introducing Azure Event Hubs

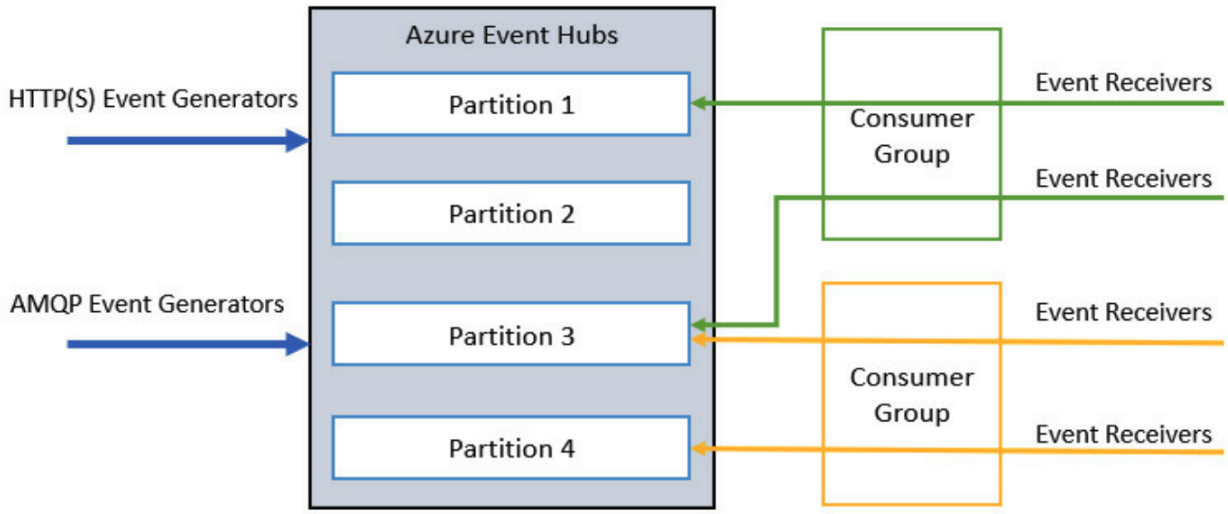
- Distributed ingestion service (fully managed)
- Ingest, store, and transfer events from input sources to consumers (typically applications)
- Azure Event Hub architecture
    - HTTP or AMQP Event Generators
    - Azure Event Hubs partitions the data
    - Consumer Groups receive and consume events

Event Hubs Partitions

- Event Hubs distributes incoming data across partitions
- Partitions help with horizontal scaling
    - Allows multiple users to read data in parallel

Event Hubs Consumer Groups

- Consumers access partitions via Consumer Groups
- Independently process the event data



Introducing Azure Streaming Analytics (ASA)

- Azure's primary stream processing engine (fully managed)
- Processes large volumes of data and used to perform analytical functions to derive quick insights
- ASA jobs typically consists of three stages
    - Read events from ingestion services
    - Process data and generate insights (transformations)
    - Write data to analytical stores

Introducing Spark Streaming

- Apache Spark can be used to build streaming solutions
- Spark streaming jobs consists of the same three stages
    - Read events from ingestion services
    - Process data and generate insights (transformations)
    - Write data to analytical stores

Developing a Stream Processing Solution

- Reference pages 275-290 for detailed walkthrough with screenshots

Develop a Streaming Solution Using Event Hubs and ASA

- Create an Event Hub instance

- Create an Event Hub within the workspace
- Create an ASA instance (Stream Analytics job)
- Link Event Hub connection string to ASA Inputs
- Link Power BI to ASA Outputs
- When events are flowing, we'll be able to view events as they arrive in the Event Hub overview page
- Read and publish data in ASA overview page
    - ASA will continuously read the input Event Hub stream, process the data, and publish to Power BI

ASA IoT Hubs

- ASA can similarly work with IoT Hub as the ingestion service
- ASA can be deployed in two modes
    - Cloud mode
    - Edge mode (ASA runs directly on the IoT devices, processing and sending events to the IoT Hub)

Develop a Streaming Solution Using Event Hubs and Spark

- Create an Event Hub instance
- Create an Event Hub within the workspace
- Create a Spark cluster
- Process and transform data in a Spark notebook

Processing Data Using Spark Structured Streaming

- Structured streaming is a feature in Apache Spark where the incoming stream is treated as an unbounded table
- Spark will process the data in frequent intervals
- There are three writing modes for the output of Structured Streaming
    - Complete mode: Entire output is written to the sink
    - Append mode: New records are appended to the sink
    - Update mode: Rows that have changed are updated to the sink
- In the above code example, the write stream is written in Delta format
    - Delta is a storage layer that can be run on top of a Data Lake
    - Enhances the Data Lake by supporting the following features;
        - ACID transactions

        - Atomicity, Consistency, Isolation, and Durability
    - Updates
    - Deletes
    - Unified batch, interactive, and streaming systems via Spark

Monitoring for Performance and Functional Regressions

- Event Hubs: The metrics tab provides metrics that can be used and plotted for monitoring
- ASA: Overview page provides metrics for monitoring
    - Alerts can be configured on top of these metrics
    - Once an alert is configured, we can configure an action group to determine an action when an alert is triggered
- Spark: Built in graphs continuously display metrics, used to monitor Spark jobs

Processing Time Series Data

- Time series data is data recorded continuously over time
- Mostly used for analyzing historic trends and forecasting
- Perfect candidate for real-time processing
- Time Series important concepts;
    - Timestamps
    - Windowed Aggregates
    - Checkpointing or Watermarking
    - Replaying

Types of Timestamps

- Event Time: When the event occurred
- Processing Time: When the event was processed

Windowed Aggregates

- Time series events are unbounded (no well-defined end time)
    - Events need to be processed in small batches (windows of time)
- Common windowing mechanisms include
    - Tumbling Windows

- Hopping Windows
- Sliding Windows

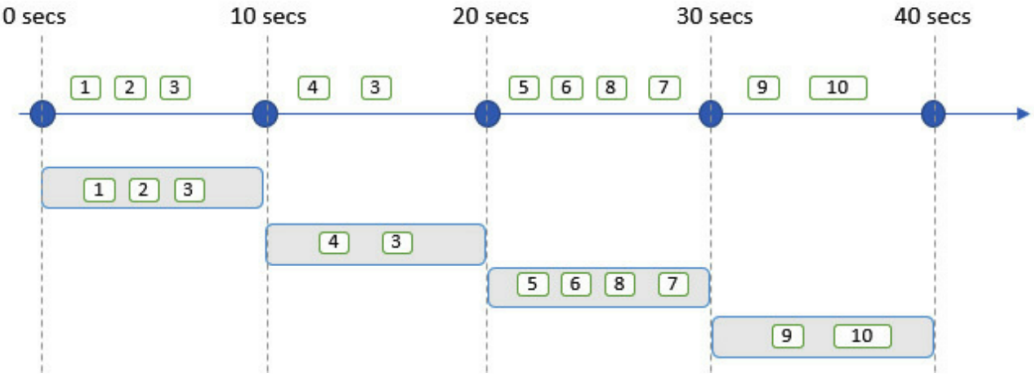
Checkpointing or Watermarking

- Process of keeping track of the last event timestamp that was processed by the stream
- Ensures we start from the previously stopped place in case of outages, processing delays, system updates, etc.

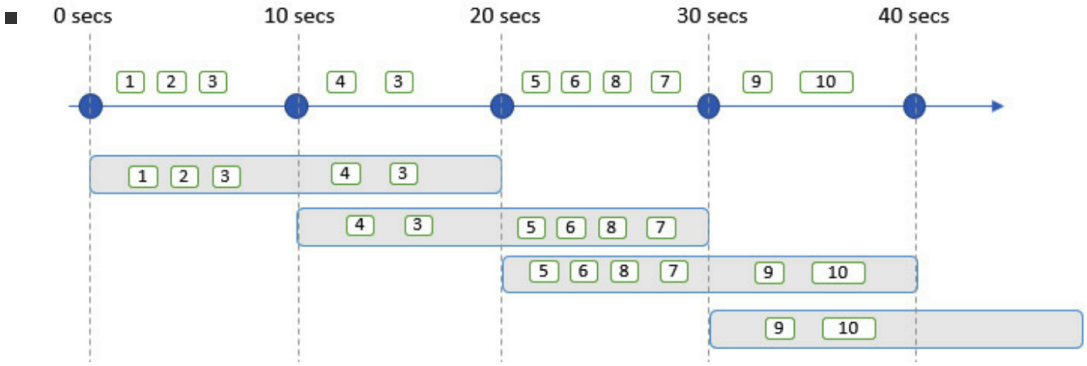
Replaying Data From a Previous Timestamp

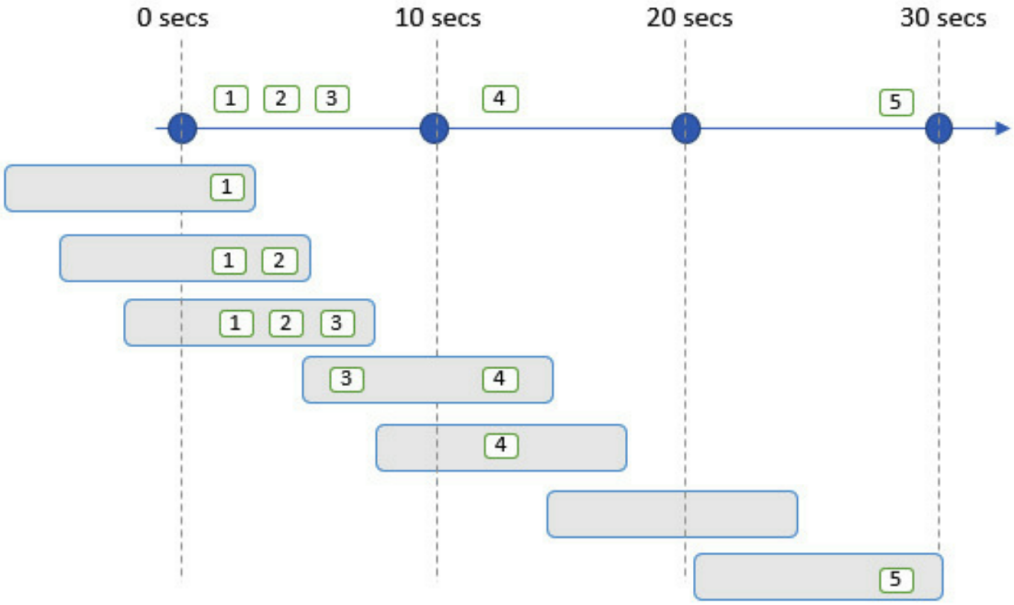
- Might be required to reprocess older data
- Event Hub provides a feature that allows us to replay events

Designing and Creating Windowed Aggregates

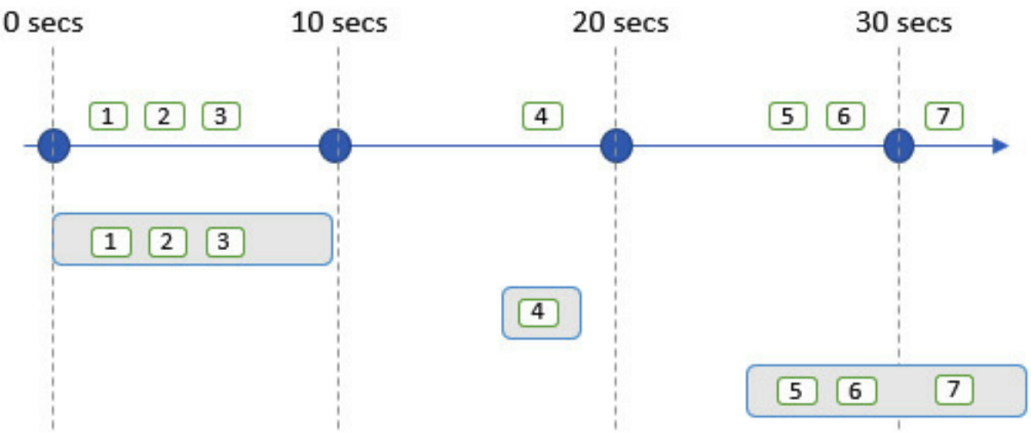
- ASA supports five types of windows
  - Tumbling windows
    - Non-overlapping, same size
    - 

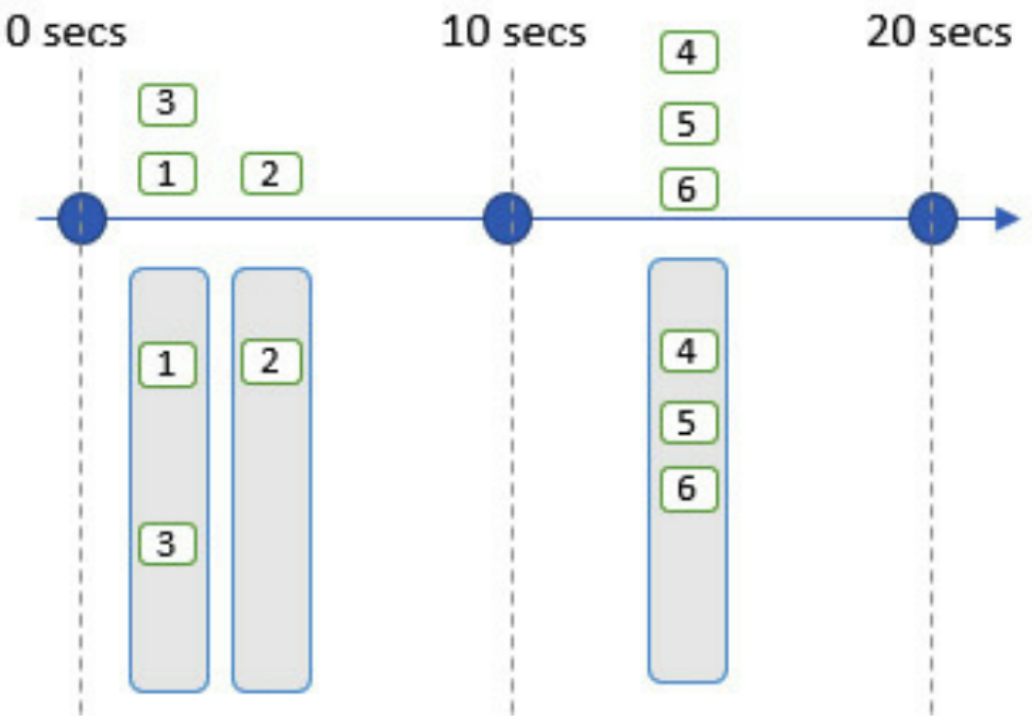
  - Hopping windows
    - Overlapping tumbling window, each window has a fixed size overlap

- 

  - Sliding windows
    - Fixed size, window only moves forward when event is added or removed
    - 

  - Session windows
    - Non-fixed size, maximum window size and timeout duration specified
    - Grabs as many events as possible during the maximum window size
    - If no events, waits for the timeout duration and closes window

- 

  - Snapshot windows
    - Not really a windowing technique
    - Simply used to get a snapshot of events at a particular time
    - 

Configuring Checkpoints/Watermarking During Processing

Checkpointing in ASA

- ASA does internal checkpointing without the need for users to do so explicitly
- Checkpointing is used for job recoveries during

    - System upgrades
    - Job retries
    - Node failures
- During ASA service upgrades, checkpoints are not maintained

Checkpointing in Event Hubs

- Refers to process of marking the offset within a stream or partition to indicate the point up to where the processing is complete
- Responsibility of the event consumer process
- Expensive operation, better to checkpoint after a batch or event processing
- Main idea is to have a restart point

Checkpointing in Spark

- Spark Structured Streaming takes care of checkpointing internally
- Ensures an event is delivered exactly once
    - Accomplished by checkpointing and write-ahead logs

Replaying Archived Stream Data

- Event Hubs stores up to 7 days of data
- Can be replayed using Event Hub consumer client libraries
- Offsets and timestamps can be specified

Transformation Using Streaming Analytics

- Common theme with streaming queries is the need for windowed aggregations for transformations
- COUNT and DISTINCT transformations used to count distinct events in a time window
- CAST transformations used to convert data types
- LIKE transformations used to perform string matching using patterns

Handling Schema Drifts

- Schema drift refers to changes in schema over time due to changes happening in event sources

- Examples include
  - New columns
  - Delete columns
  - Updated fields

Handling Schema Drift in Event Hubs

- Event Hubs provides a feature called Azure Schema Registry to handle schema drift
- Provides a central repository to share schemas between event publishers and consumers

Handling Schema Drift in Spark

- Azure Databricks Delta Lake provides a feature called Schema Evolution to handle schema changes over time
- When enabled, it automatically adapts to the new schema when columns are added

Processing Across Partitions

- Event Hubs can distribute incoming events into parallel streams (partitions)
- Each partition stores the event data and metadata
- Partitioning helps in scaling real-time processing
  - Increases the parallelism by providing multiple input streams for downstream processing engines
- Improves availability by redirecting events to healthy partitions when others fail

Processing Data Across Partitions

- EventHubConsumerClient (class) can be used to read events from all partitions with load balancing and checkpointing
- If a partition is not specified, it will automatically read across all partitions in the specified consumer group

Processing Within One Partition

- EventHubConsumerClient (class) can also be used to process data within a single partition

- Specify the partition ID in the client.recieve() call

Scaling Resources

Scaling in Event Hubs

- Two ways Event Hubs supports scaling
  - Partitioning
  - Auto-inflate
- Auto-inflate adds more throughput as usage increases
  - Enabled by setting Enable Auto-inflate to True when creating a Namespace in Event Hub

Scaling in ASA

- ASA clusters can be scaled directly from the Azure portal
  - Scale tab
  - Streaming Unit toggle
- Streaming Units are a measure of streaming capacity
- Azure recommends starting with six SU's and modifying based on the SU% utilization metric
  - For queries not using the PARTITION BY transformation
- ASA does not offer an internal auto scale mechanism
  - Can be accomplished with Azure Automation service to monitor ASA metrics and trigger a scale up or down

Scaling in Spark

- Auto-scaling reviewed in Ch 9, Scaling Resources section

Handling Interruptions

- Event Hubs and ASA provide options to handle service interruptions with availability zones
- Availability zones are physically isolated locations in Azure that help applications become resilient to local failures and outages

- Services that support availability zones deploy applications to all locations within the availability zone to improve fault tolerance
    - Both events and metadata are deployed
- Service upgrades are always done one after the other for the locations
- Event Hubs and ASA will have availability zones enabled if a location that supports them is selected

Designing and Configuring Exception Handling

- The EventHubsException object contains the following information
    - IsTransient: Indicates whether the exceptions can be retried
    - Reason: Indicates the actual reason for the exception

Upserting Data

- Upserting refers to the INSERT or UPDATE activity in a database or any supporting analytical data store
- ASA has two different behaviors based on the specified compatibility level (Compatibility Level option in ASA Job, ie Version)
    - Levels 1.0 and 1.1: ASA does a property level INSERT or UPDATE within the document
    - Level 1.2 and onward: ASA does an INSERT or replace (UPDATE) document operation

Designing and Creating Tests for Data Pipelines

- Covered in Ch 9: Designing and Developing a Batch Processing Solution
    - Section: Designing and Creating Tests for Data Pipelines

Optimizing Pipelines for Analytical or Transaction Purposes

- Covered in Ch 14: Optimizing and Troubleshooting Data Storage and Data Processing
    - Section: Optimizing Pipelines for Analytical or Transactional Purposes

Summary

That brings us to the end of this chapter. This is one of the important chapters both from a syllabus perspective and a data engineering perspective. Batch and streaming solutions are fundamental to building a good big data processing system.

So, let's recap what we learned in this chapter. We started with designs for streaming systems using Event Hubs, ASA, and Spark Streaming. We learned how to monitor such systems using the monitoring options available within each of those services. Then, we learned about time series data and important concepts such as windowed aggregates, checkpointing, replaying archived data, handling schema drifts, how to scale using partitions, and adding processing units. Additionally, we explored the upsert feature, and towards the end, we learned about error handling and interruption handling.

You should now be comfortable with creating streaming solutions in Azure. As always, please go through the follow-up links that have been provided to learn more, and try the examples yourself to understand the nitty-gritty details of these technologies.

In the next chapter, we will learn how to manage batches and pipelines.

Chapter 11: Managing Batches and Pipelines

Triggering Batches

- Azure Batch jobs can be triggered with Azure functions
- Azure Functions are a serverless service that helps build cloud applications with minimal code and no hosting or maintenance
- A Trigger defines when and how to invoke an Azure Function
- Azure Functions support the following triggers
    - Timer triggers
    - HTTP triggers
    - Blob triggers
    - Event Hub triggers, etc.

Creating an Azure Function and Trigger

- Create a Functions service
- Select a trigger template and create a Function
- Define the trigger in the Code + Test tab
    - C#, Java, Python, or PowerShell

Handling Failed Batch Loads

- Azure Batch can fail due to four error types
    - Pool errors
    - Node errors
    - Job errors
    - Task errors
- Settings can be updated and error logs can be viewed from the following APIs

```
# Update Timeout
POST {batchUrl}/pools?timeout={timeout}&api-version=<date>

# View Error Logs
GET {batchUrl}/pools/{poolId}/nodes/{nodeId}?api-version=<date>
```

Pool Errors

- Occur due to infrastructure issues, quota issues, or timeout issues
- Sample pool errors
    - Insufficient Quota
    - Insufficient Resources in VNet
    - Short Timeouts

Node Errors

- Occur due to hardware issues or failures in job setup activities
- Sample node errors
    - Start task failures: Reference Task Execution Result and Task Failure Information properties in API GET call

    - Application download failures: Reference Compute Node Error property in API GET call
    - Nodes going into a bad state: Reference Compute Node Error property in API GET call

Job Errors

- Typical job errors occur due to
    - Job constraints: Reference Job Execution Information property in API GET call
    - Errors in job preparation or job release tasks: Reference Job Scheduling Error property in API GET call

Task Errors

- Typical task errors occur due to
    - Tasks command fails and returns a non-zero exit code
    - Resource file download issues
    - Output files upload issues
- All errors can be referenced in Task Execution Information property from API GET call

Validating Batch Loads

- Batch jobs are usually run in ADF
- ADF provides activities for validating job outcomes

Validating Batch Loads in ADF

- Add Validation activity (used to check for a file's existence before proceeding)
- Add Get Metadata activity to get more information about the file
- Use If Condition activity to write business logic based on the metadata

Scheduling Data Pipelines in ADF/Synapse

- Process of defining when and how a pipeline needs to be started
- Both ADF and Synapse Pipelines have a Add Trigger in the Pipelines tab that can be used to schedule pipelines

    - Options to Trigger Now, or New/Edit (add or edit an existing trigger)
- Pipeline services support four types of triggers
    - Schedule Trigger: Trigger pipeline once or regularly based on clock time
    - Tumbling Window Trigger: Trigger pipeline based on periodic intervals while maintaining state
    - Event-Based Trigger: Trigger pipeline based on storage event
    - Custom Trigger: Trigger pipeline based on events from Azure Event Grid
- View pipeline triggers from the Monitoring tab

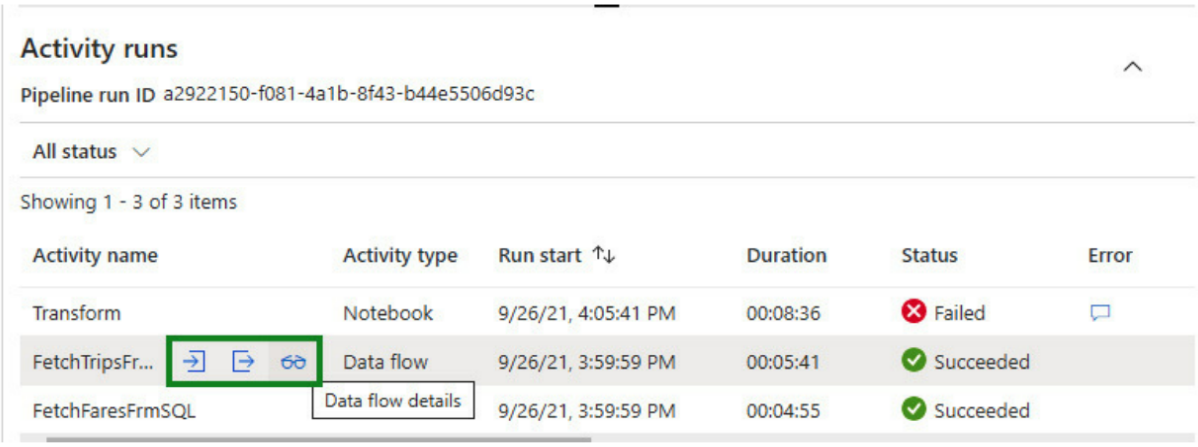
Managing Data Pipelines in ADF/Synapse

Manage Tab

- Configure linked services, integration runtimes, triggers, Git, etc.
- Integration runtimes (IR) refer to compute infrastructure used to run pipelines and data flows
    - These are the actual machines that run jobs behind the scenes
    - IRs handle
        - Running data flows
        - Copying data across public and private networks
        - Dispatching activities to services
    - Support for three types of IRs
        - Azure Integration Runtime: Supports connecting data stores and compute services across public and private endpoints
        - Self-Hosted Integration Runtime: Supports copying data between on-premises clusters and Azure services
        - Azure SSIS Integration Runtime: Supports lift and shift use cases

Monitoring Tab

- Pipelines can be triggered manually or automatically (using triggers)
- In-progress runs can be cancelled
- Failed runs can be Rerun
- Click on a pipeline run to look at the details of each flow
    - Click on spectacles icon for run statistics



Managing Spark Jobs in a Pipeline

- Involves two aspects
  - Managing the attributes of the pipeline's runtime that launches the Spark activity
  - Managing Spark jobs and configurations
- A Spark job file can be provided in the Script/Jar tab of the Spark activity
  - Entire Spark pipeline will be run upon trigger

Implementing Version Control for Pipeline Artifacts

- ADF and Synapse Pipelines save pipeline details in internal stores by default
- These stores don't provide options for collaboration or version control
- Every time Publish All is selected, latest changes are saved within the service (lose previous version)
- ADF and Synapse Pipelines provide a Set Up Code Repository button on the home screen (alternatively configure in the manage tab)
  - Support for Azure Dev Ops and GitHub version control
  - In both, ADF creates a new branch called adf_publish
    - Unable to make changes to branch but can merge vis pull requests
  - When configured, Publish will store pipeline activities in the Git repository

Summary

With that, we have come to the end of this small chapter. We started by learning how to trigger Batch loads, how to handle errors and validate Batch jobs, and then moved on to ADF and Synapse pipelines. We learned about setting up triggers, managing and monitoring pipelines, running Spark pipelines, and configuring version control in ADF and Synapse Pipelines. With all this knowledge, you should now be confident in creating and managing pipelines using ADF, Synapse Pipelines, and Azure Batch.

This chapter marks the end of the Designing and Developing Data Processing section, which accounts for about 25-30% of the certification goals. From the next chapter onward, we will next move on to the Designing and Implementing Data Security section, where we will be focusing on the security aspects of data processing.

Chapter 12: Designing Security for Data Policies and Standards

Designing and Implementing Dat Encryption (At Rest and In-Transit)

- Data needs to be encrypted at rest and in transit
- Typically required for regulation compliance

Encryption at Rest

- Process of encrypting data before writing to disk and decrypting when requested by applications
- Protects data from
    - Physical disk theft
    - Data retrieval from lost disks
    - Unauthorized access by cloud company employees, etc.
- Data cannot be retrieved when encrypted (will appear as gibberish)

Encryption at Rest for Azure Storage

- Provided by default (cannot be disabled)

- Options to use Azure keys or Customer-Managed Keys (CMKs)
    - Enabled from the Encryption tab in Azure Storage
    - CMKs need to be stored in Azure Key Vault
        - Azure Key Vault stores passwords, secret keys, access keys, etc.

Encryption at Rest for Azure Synapse SQL (Applies to All Azure SQL Tech)

- Accomplished using a feature called Transparent Data Encryption (TDE)
- TDE must be manually enabled for Azure Synapse SQL (applied by default in other Azure SQL tech)
    - Enable TDE from SQL Pool screen under Transparent Data Encryption tab
- TDE encrypts the database, backup, and snapshot files
- Options to use Azure keys or Customer-Managed Keys (CMKs)
    - If using CMKs, should enable Double Encryption Using CMK when creating the Synapse workspace

Always Encrypted

- Feature provided by Azure SQL and SQL Server databases
- Encrypts selected database columns using client drivers
- Encryption keys are fetched from a secure location (Azure Key Vault) to encrypt or decrypt specified columns
- Only data owners will have access to the encrypted keys and data (used for columns with Personal Identifiable Information - PII)
- Two types of keys used for Always Encrypted
    - Column Encryption Key: Key used to encrypt and decrypt a column
    - Column Master Key: Protection key that encrypts the Column Encryption Keys
- When writing encryption code, the following options for ENCRYPTION_TYPE are important
    - DETERMINISTIC ensures client drivers generate the same encrypted value (useful for JOINs, INDEXES, AGGREGATES, etc.)
    - RANDOMIZE generates a different encrypted value each time
- There are four database permissions needed for Always Encrypted
    - ALTER ANY COLUMN MASTER KEY
    - ALTER ANY COLUMN ENCRYPTION KEY
    - VIEW ANY COLUMN MASTER KEY DEFINITION
    - VIEW ANY COLUMN ENCRYPTION KEY DEFINITION

- Operations such as SELECT INTO, UPDATE, and BULK INSERT will not work with Always Encrypted columns

Encryption in Transit

- Process of encrypting data sent over a wire (moving from one place to another over the net)
- Data movement involves
  - Read by an application
  - Replication to different zones
  - Downloading from cloud
- There are two encryption in transit protocols
  - Secure Socket Layer (SSL) which is being discontinued
  - Transport Layer Security (TLS) which is the preferred protocol
- Other ways to encrypt data in transit is to setup VPNs or Azure Express Route
  - Express Route provides a private connection from on-premises networks to the Azure cloud

Encryption in Transit for Azure Storage

- Enabled in the storage account home page under the Configuration tab
- Select a Minimum TLS Version (1.2 is recommended as of the writing of this book)

Encryption in Transit for Azure Synapse SQL

- Synapse automatically encrypts data in transit using TLS protocol

Designing and Implementing a Data Auditing Strategy

- Data auditing is the process of keeping track of activities performed in a service

Azure Storage Auditing

- Supported via Storage Analytics Logging (classic monitoring) and now Azure Monitor (newer version)
- Storage Analytics Logging can be enabled under the Diagnostic Setting (classic) tab
  - Logs are stored in a container called $logs in the storage account
- Azure Monitor is now enabled by default (Diagnostic Setting (preview) tab)

    - Collects default metrics and logs for every service and can be configured to collect additional metrics
    - Logs, metrics, and storage containers can be specified
    - There is a storage cost associated

SQL Auditing

- Azure Synapse SQL provides the option to track all database events and activities using the Auditing feature
- Enabled from the Azure SQL Auditing tab in the SQL Pool
- Auditing provides multiple storage options
    - Azure Blob Storage: Useful for storing data for compliance
    - Log Analytics: Useful for ad-hoc querying and analysis
    - Event Hub: Useful for real-time analysis

Designing and Implementing a Data Masking Strategy

- Data masking is a technique used to hide sensitive data in a SQL query result (this is not a column encryption)
- The data itself is not changed, the queries and views modify the data dynamically to mask sensitive information from non-privileged users
- Can be enabled in Synapse under the Dynamic Data Masking tab
- Can also be enabled with a T-SQL script
    - ALTER COLUMN < col name > ADD MASKED WITH (FUNCTION = '< masking function >')

Designing and Implementing Azure Role-Based Access Control for ADLS Gen2

- Azure uses and recommends the *principle of least privilege*
- PLP: Assigning the least possible privilege required to accomplish a task

Restricting Access Using Azure RBAC

- Azure Role-Based Access Control (RBAC) is an authorization system that controls who can access what resources in Azure
- RBAC has three components that must be defined to create an RBAC rule

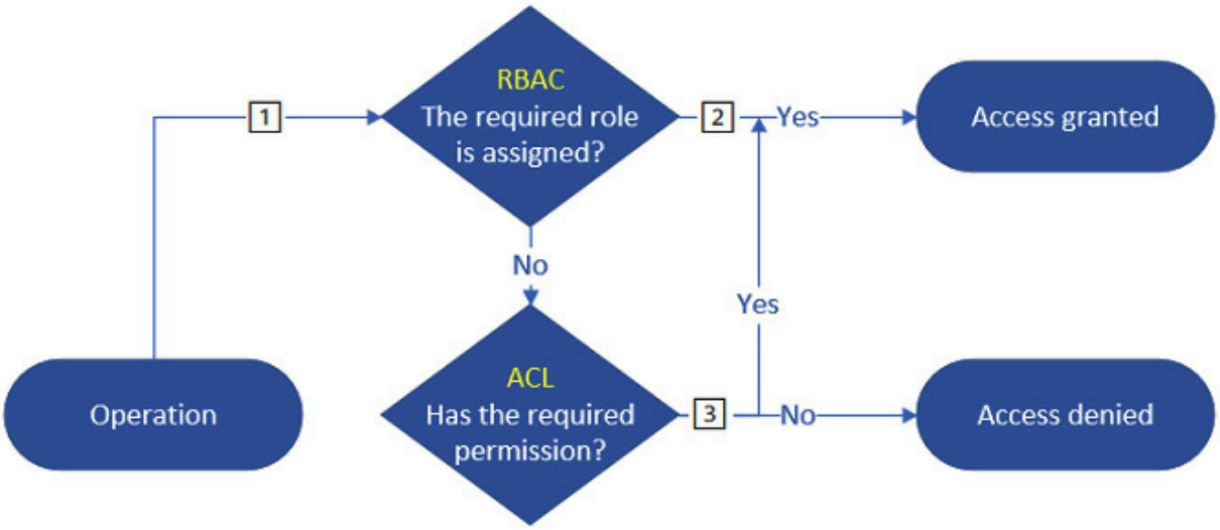
    - Security Principal: Real people or service accounts used to run services automatically
    - Role: Defines what actions can be performed by a user (Owner, Contributor, Reader, etc.)
    - Scope: Refers to all the resources where a role needs to be applied
- RBACs can be created under the Access Control (IAM) tab from the Azure Storage home page

Restricting Access Using ACLs

- Access Control Lists (ACLs) provide more fine-grained access such as who can read data from a specific directory or file
- Enabled by default in ADLS Gen2
- RBAC and ACL complement each other to provide a wide spectrum of access control
- Each directory and file in Azure Storage has an ACL
- You can assign any (or all) read, write, and execute permissions to individual security principals or groups
    - Right click on a folder or file name and select Manage ACL

How Azure Decides Between RBAC and ACLs (Conflicting Rules)

- In case of conflicts, Azure gives precedence to RBAC



Limitations of RBAC and ACL

- Restrictions on RBAC and ACL include
  - RBAC allows 2,000 role assignments per subscription
  - ACL allows up to 32 ACL entries per file and directory
    - This is a little more restrictive
    - Make sure to add groups to ACLs instead of individual users
- Azure also supports two other authentication methods
  - Shared Key Authorization
    - Shares an access key giving admin-like access to the resource
  - Shared Access Signature (SAS) Authorization
    - SASs allows you to define what actions are allowed with the key
  - Both SKA and SAS will override RBACs and ACLs

Designing and Implementing Row-Level and Column-Level Security

- Fine-grained security implementations at the row and column level
- Restricts data acess

Designing Row-Level Security

- RLS restricts access to specified rows in a table
- Similar to how a WHERE clause would filter a result set
- Achieved by creating security policy rules that reside in the database
  - Regardless how data is accessed, restrictions will be enforced
- Creating a RLS rule in T-SQL (Pg 368)
  - Create a security schema
  - Create a function with the restriction logic
  - Create a security policy with previously defined function
  - Test the policy

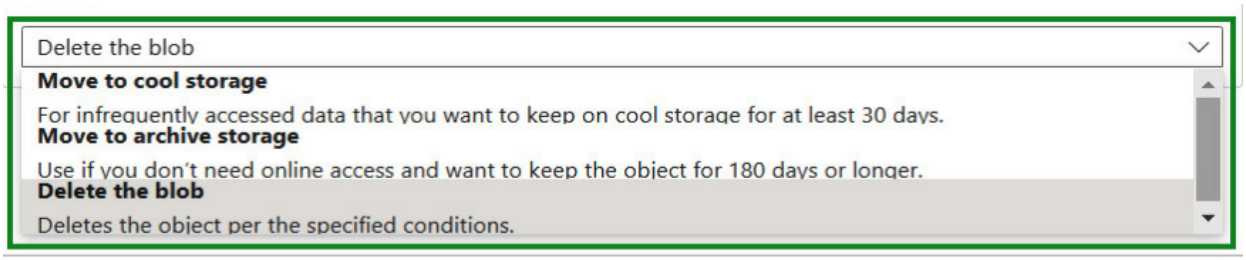
Designing Column-Level Security

- Similar to data masking feature
  - Instead of masking values, entire columns are restricted
- Useful when only a small subset of data in a table is sensitive

- Eliminates the need to split and store sensitive data seperately
- Achieved by creating security policy rules that reside in the database
    - Regardless how data is accessed, restrictions will be enforced
- Creating CLS rule in T-SQL (Pg 370)
    - Use the GRANT command to grant permissions to specified columns in a table to a user
    - Test the policy

Designing and Implementing a Data Retention Policy

- Azure Storage has a Data Lifecycle Management screen with options to move data to Cool or Archive tiers, or delete the blob after a specified time



Designing to Purge Data Based on Business Requirements

- Data purging refers to the process of safely deleting data as per business requirements
- Two ways to purge data in ADLS Gen2 (storage locations are overwritten to prevent deleted data retrieval)
    - Azure Storage Life Cycle Management (explained above)
    - ADF Delete Activity
- Purging data in Synapse SQL is done using the TRUNCATE command
    - More efficient than DELETE because it does not generate transaction logs
    - Cannot be combined with a WHERE clause
    - A DELETE statement can be used with a WHERE clause if needed
    - Ideally, archived data is stored separately and the entire table can be truncated

Managing Identities, Keys, and Secrets Across Different Data Platform Technologies

- Two main technologies used in Azure
  - Azure Active Directory (AAD)
  - Azure Key Vault

Azure Active Directory

- AAD is Azure's identity and access management service
- Supports managing users, groups, service principals, etc.
- Managed in the Azure Portal (search for AAD)
  - Manage Users, Groups, External Identities, Roles and Administrators

Managed Identities

- Feature supported by AAD
- Managed Identities are assigned to instances of Azure services when created
- Managed Identity ID can be found in the service overview page

Azure Key Vault

- Used to store keys, secrets, and certificates securely
- Encrypts and decrypts using 256-bit AES encryption
- Provides the following functionalities
  - Key Management
  - Secrets Management
  - Certificate Management
- Centralized service, every time a secret or key needs to be changed it only has to be updated in Key Vault
- In order to use Key Vault in Synapse, the Key Vault has to be added as a linked service
  - Subsequent linked services will have option to choose secrets directly from Key Vault

Access Keys and Shared Access Keys

- Access Keys give complete access to all resources in a storage account

- Shared Access Signature (SAS) Keys grant restricted access
- Types of SAS Keys
  - Service SAS
  - Account SAS
  - User Delegation SAS

Implementing Secure Endpoints (Private and Public)

- Any service created in Azure without configuring a VNet uses public endpoints (default)
- Private endpoints are part of the Private Link service
  - Private endpoints refer to a network interface that uses the private IP address from your VNet
  - Private Link service refers to the overall service that comprises the private endpoints and private networks over which the traffic traverses
- Private endpoints can be configured for services manually or managed by Azure
  - To manually create, you need to create a VNet and a Private Link service where you can create a private endpoint for a resource (Synapse in this example)
  - Once created, the Synapse workspace will only be accessible within the VNet you created
- Azure provides a Managed Virtual Network and Managed Endpoints options
  - When you create a Managed VNet, Azure creates the necessary VNets and private endpoints, firewall rules, subnets, etc
  - Managed VNets and Endpoints are no different than manually created ones, except that they are managed and less error-prone

Implementing Resource Tokens in Azure Databricks

- Azure Databricks provides access tokens called Personal Access Tokens (PATs)
- Used to authenticate Azure Databricks APIs
- When generated, the token is displayed in a pop up screen and most be copied and stored as it will not be accessible once closed
- Maximum number of PATs per Azure Databricks workspace is 600
- Similar to PATs, AAD tokens can also be used for authorization

Loading a DataFrame with Sensitive Information

- When this book was written, Spark does not offer techniques such as data masking and row/column level security
- The Fernet Library can be used in Spark to encrypt and decrypt text
- The encryption key would be needed to decrypt an encrypted column

Writing Encrypted Data to Tables or Parquet Files

- Covered in a previous example (reference code repository)

```
# Write Encrypted Data to Tables
df.write.format("delta").mode("overwrite").
    option("overwriteSchema", "true").
    saveAsTable("PIIEncryptedTable")

# Write Encrypted Data to Parquet Files
encrypted.write.mode("overwrite").parquet("abfss://path/to/store")
```

Designing for Data Privacy and Managing Sensitive Information

- Organizations that handle sensitive information are bound legally to secure data
- Securing sensitive data is also important to maintain an organizations reputations
- Azure security standards recommend the following techniques to keep sensitive data safe
    - Identifying and classifying sensitive data
        - Synapse SQL portal provides a feature that automatically suggests sensitive columns
        - Data Discovery and Classification tab
    - Protecting the sensitive data using techniques such as
        - Storing sensitive data in different accounts, partitions, or folders
        - Restricting access using RBAC and ACLs
        - Encrypting data at rest and in transit
        - Data masking
        - Row and column-level security

    - Monitoring and auditing the consumption of sensitive data

Microsoft Defender

- Another service option for security and threat management
- Provides tools and services required to monitor, alert, and mitigate threats
- Natively integrated in most Azure services, enabled easily
- Microsoft Defender for Storage helps identify threats such as
    - Anonymous access
    - Malicious content
    - Compromised credentials
    - Privilege abuse, etc.
- Microsoft Defender for SQL helps identify threats such as
    - SQL injection
    - Brute-force attacks
    - Privilege abuse, etc.

Summary

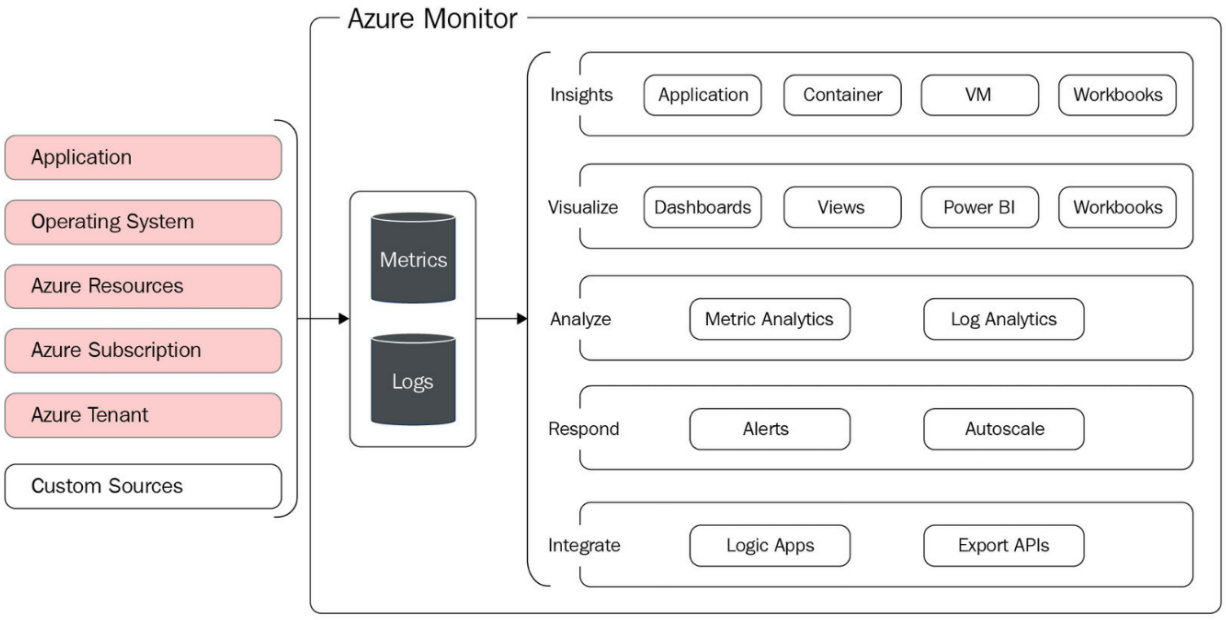
With that, we have come to the end of this chapter. This chapter is one of the lengthiest chapters but luckily not so complicated. We started with designing the security requirements of our IAC example and then used that as our guide to explore the various security and compliance topics. We learned about encryption at rest and transit, enabling auditing for Azure Data Lake Storage and Synapse SQL, implementing data masking, RBAC and ACL rules, row- and column-level security, and had a recap on data retention and purging. After that, we continued with topics such as AAD and Key Vault to manage keys, secrets, and certificates, learned about secure endpoints, and details of tokens and encryption in Spark. Finally, we wrapped it up with concepts to be followed to design data privacy. You have now covered more or less all the important topics in handling security and privacy. You should now be able to design and implement a complete security and privacy solution on top of Azure Data Lake.

We will be exploring monitoring and optimization techniques for data storage and data processing in the next chapter.

Chapter 13: Monitoring Data Storage and Data Processing

Implementing Logging Used by Azure Monitor

- Azure Monitor is the service we use to monitor
  - Infrastructure
  - Services
  - Applications
- Azure Monitor records two types of data
  - Metrics
    - Numerical values that describe an entity or an aspect of a system at different points of time
    - Stored in time-series databases
    - Can be aggregated for alerting, reporting, and auditing purposes
  - Logs
    - Text details of what is happening in the system
    - Usually event-driven
- Azure Monitor is an independent service
  - Can aggregate logs from different services to give a global perspective
- Log Analytics Workspace is a sub-service of Azure Monitor
  - Specializes in processing and exploring log messages
  - Accessible from the Logs tab of any Azure service
  - Supports Kusto Query Language (KQL) to analyze log data
  - May incur additional storage and compute costs
    - Alternative is to store logs in regular Azure Storage options or use Event Hub for real-time log processing



Logging Data in a Log Analytics Workspace (Pg 402)

- Create a Log Analytics Workspace
- Navigate to the Azure Monitor service and click on the Activity Log tab
- Click on Diagnostic Settings in the Activity Log tab
- Select which logs to send to Log Analytics Workspace
- In the Log Analytics Workspace, an AzureActivity table is created in the Logs tab
- All activity logs will be stored here and can be queried using KQL

Configuring Monitoring Services

- Azure Monitor is created as soon as an Azure resource is created
- Basic metrics and logs are recorded by default
- Additional configurations can be set within Azure Monitor (such as sending logs to Log Analytics)
- Monitoring can be configured at multiple levels
    - Application monitoring
    - Operating system (OS) monitoring
    - Azure resource monitoring
    - Subscription-level monitoring
    - Tenant-level monitoring

Resource, Subscription-Level, and Tenant-Level Monitoring

- Metrics and logs are generated by default
- Moving the data to a log destination needs to be set in the Diagnostic Settings in Azure Monitor
- Log destination options
    - Log Analytics Workspace
    - Event Hubs
    - Storage Account

Application and OS Monitoring

- An Azure Monitor Agent (AMA) or Log Analytics Agent needs to be installed in order to start collecting metrics and logs

Options for Monitoring Azure Storage

- Metrics available in the Metrics tab of Azure Storage
    - Ingress
    - Egress
    - Blob Capacity, etc
- Metrics available in the Insights tab of Azure Storage
    - Failures
    - Performance
    - Availability
    - Capacity
- Metrics available in Azure Monitor for Azure Storage
    - Storage service status
    - Transactions
    - Transactions Timeline, etc

Understanding Custom Logging Options

- Custom Logs option in Azure Monitor
- Collects text-based logs not part of the default logs collected such as
    - System logs
    - Event logs in Windows

    - Event logs in Linux
- Host machine must have the Log Analytics Agent installed to configure

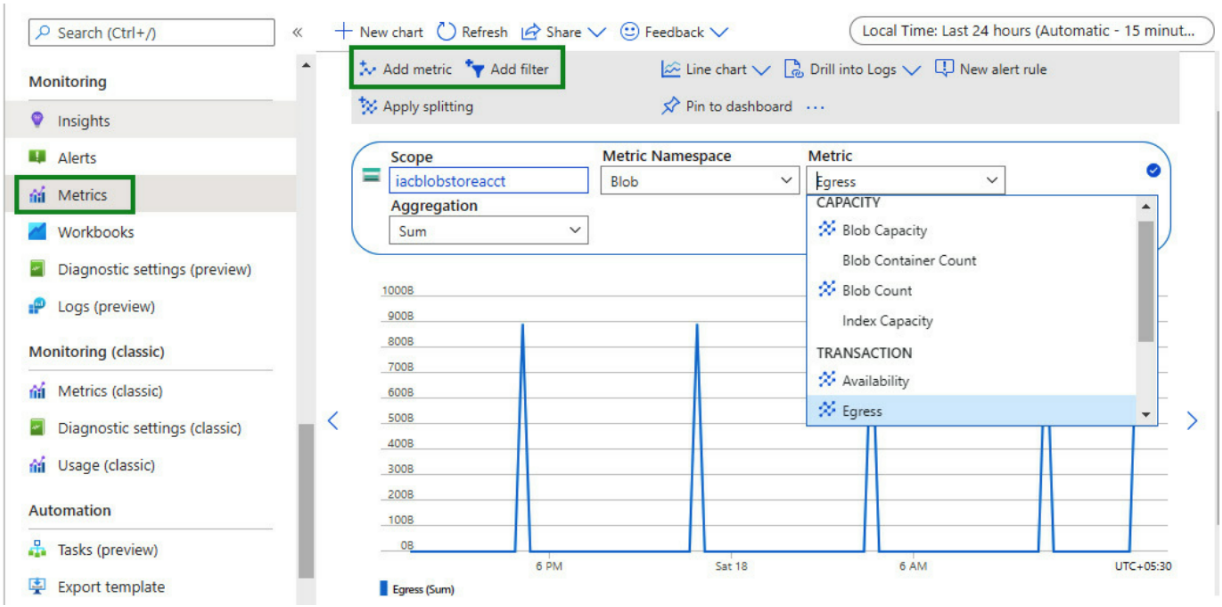
Configuring Custom Logs

- Navigate to Log Analytics Workspace, Custom Logs tab
- Click on Add Custom Log
- Upload a sample log file
- Enter OS type and path of log files
- Enter custom log identifying details and create

Interpreting Azure Monitor Metrics and Logs

Azure Monitor Metrics

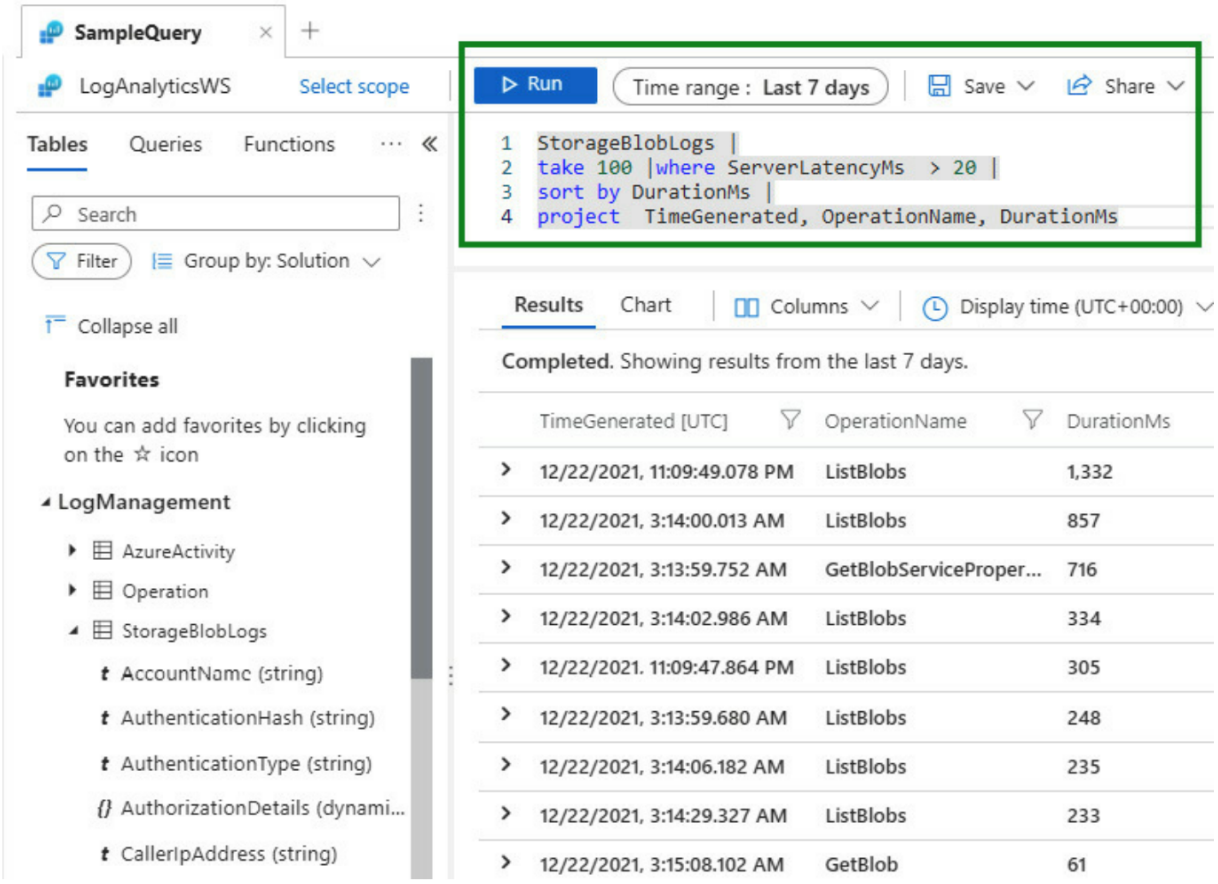
- Metrics can be aggregated based on Sum, Avg, Min, and Max
- Additional metrics can be overalyed
- Metrics can be filtered
- Data is available for 30 days using the metrics explorer in Azure Monitor
    - Will have configure long-term storage



Azure Monitor Logs

- Logs will have to be analyzed in a Log Analytics Workspace (detailed above)
- Logs will be stored in tables under LogManagement in the Logs tab
- KQL is used to query and analyze log data
    - KQL queries consists of statements delimited by semicolons
    - Pipes (|) are used signify steps

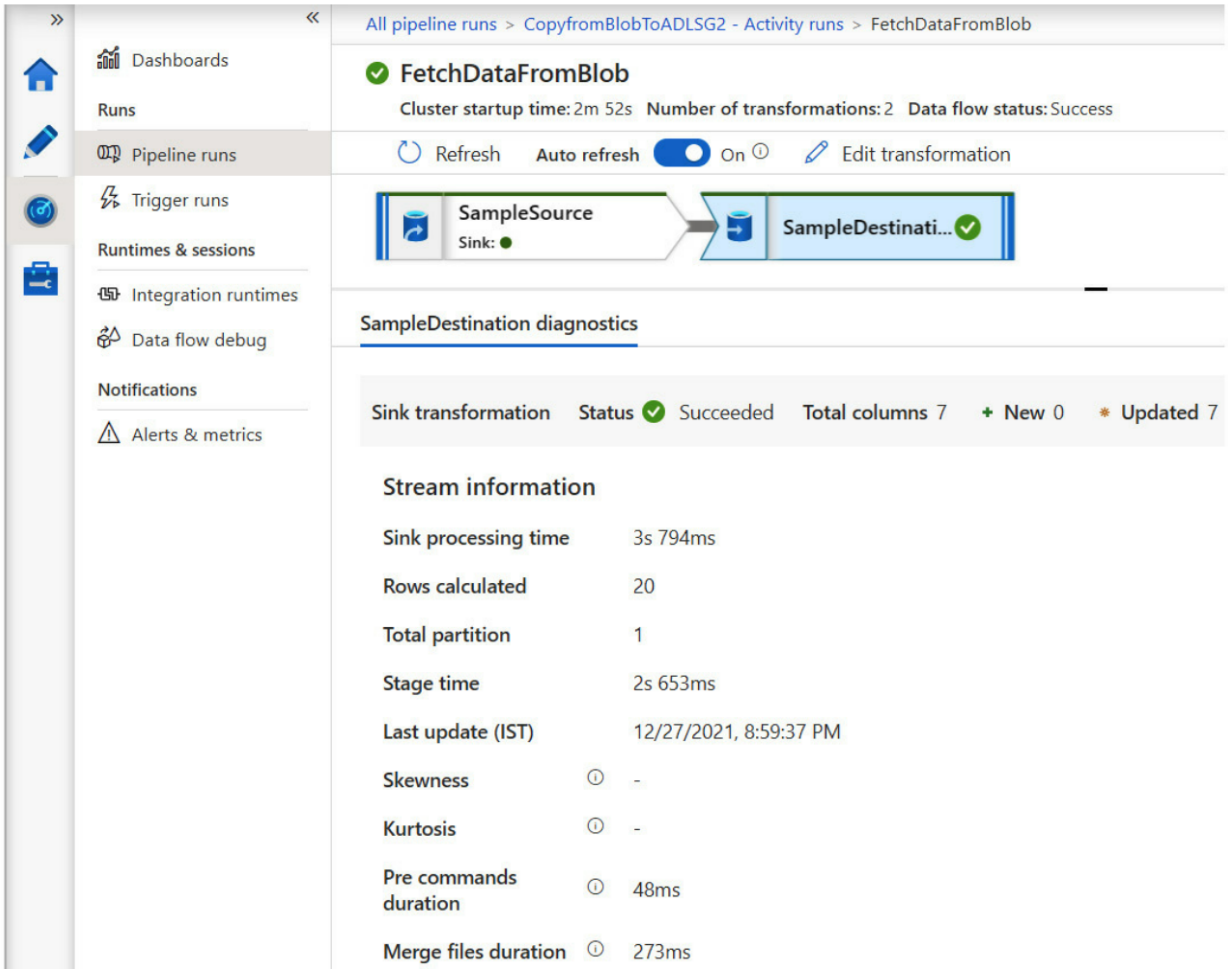
```
StorageBlobLogs |
TAKE 100 |
WHERE ServerLatencyMs > 20 |
SORT BY DurationMs |
PROJECT TimeGenerated, OperationName, DurationMs
```



Measuring the Performance of Data Movement

- ADF provides performance metrics under the Monitoring tab

- In the monitoring tab, we can click on the Pipeline Runs tab to see details of each piepline
- Clicking on any of the steps will show diagnostic details



Monitoring Data Pipeline Performance

- Similar to above, if we hove over a pipeline in the Pipeline Runs tab, we can click on a Consumption icon for details
- There is a List and Gnatt display option for details

ADF stores pipeline execution details and metrics for 45 days. Longer storage requirements will require a Log Analytics Workspace.

Monitoring and Updating Statistics About Data Across a System

- Collecting statistics is the process of collecting metadata about data
- Statistics are important for query optimization
- They can be used as additional inputs by the SQL engine to optimize query plans
- The Synapse SQL pool engine uses cost-based optimizers (CBOs)
    - CBOs choose the least expensive query plan from a set of query plans

Creating Statistics for Synapse Dedicated Pools

- Statistics can be enabled for Synapse Dedicated Pools using the below query
    - Excluding temporary or external tables
- Once AUTO_CREATE_STATISTICS is ON, any of the following statements will trigger statistics for the columns involved in the query
    - SELECT
    - INSERT-SELECT
    - CTAS (CREATE TABLE AS)
    - UPDATE
    - DELETE
    - EXPLAIN

```
ALTER DATABASE [DW_NAME] SET AUTO_CREATE_STATISTICS ON;
```

- Statistics on demand can be generated using the below query
- SAMPLE default is 20%

```
CREATE STATISTICS [statistics_name]
    ON [schema_name].[table_name]([column_name])
    WITH SAMPLE 40 PERCENT;

-- Fullscan Option
CREATE STATISTICS [statistics_name]
    ON [schema_name].[table_name]([column_name])
    WITH FULLSCAN;
```

Updating Statistics for Synapse Dedicated Pools

- Statistics can also be periodically updated

- It's a good practice to update statistics every fresh data load

```
UPDATE STATISTICS [schema_name].[table_name]([stat_name]);
```

Creating Statistics for Synapse Serverless Pools

- The concept of statistics is the same for Dedicated and Serverless pools
- For Serverless pools, the auto-creation of statistics option is on by default for Parquet files only
- Since we deal with external tables in Serverless pools, we have to manually create statistics for them using the following query
    - Only single column statistics are available for external tables

```
CREATE STATISTICS [statistics_name]
    ON { external_table } ( column )
    WITH
        { FULLSCAN |
        [ SAMPLE number PERCENT ] }
        , { NORECOMPUTE } ;
```

Updating Statistics for Synapse Serverless Pools

- Updating statistics in Serverless pools require dropping existing statistics and recreating them using the following query

```
DROP STATISTICS [OLD_STATISTICS_TABLE_NAME];

CREATE STATISTICS [statistics_name]
    ON [NEW_STATISTICS_TABLE_NAME] (STATENAME)
    WITH FULLSCAN, NORECOMPUTE;
```

Note on Statistics

- Statistics collection might cause a slight performance degradation if statistics are missing for columns in a query
- Once statistics are generated, future queries will typically run much faster

Measuring Query Performance

For measuring the performance of any SQL-based queries, it is recommended to set up the Transaction Processing Performance Council H or DS (TPC-H or TPC-DS) benchmarking suites and run them on a regular basis to identify any regressions in the platform. TPC-H and TPC-DS are industry-standard benchmarking test suites.

Monitoring Synapse SQL Pool Performance

- The Metrics tab in Synapse SQL Pool provides metrics for
  - Data Warehouse Units (DWUs) limits, usage, and usage %
  - Memory usage %
  - Queued queries, etc
- Does not provide query performance or query wait time details
  - Those can be found by querying system tables or using Query Store

Querying System Tables

- The following system tables can be used to monitor query performance
  - sys.dm_pdw_exec_requests
    - Contains all current and recently active requests in Azure Synapse Analytics
    - Contains details such as
      - total_elapsed_time
      - submit_time
      - start_time
      - end_time
      - command
      - result_cache_hit, etc
  - sys.dm_pdw_waits
    - Contains details of the wait states in a query, including locks and waits on transmission queues.

Using Query Store

- Azure SQL and Synapse SQL support a feature called Query Store
- Used for query performance monitoring and tuning
- Query Store is used to store history of queries, plans, and runtime statistics
- This data can be used to aid in
  - Query optimization

    - Identifying query regressions
    - Monitoring trends in resource utilization (such as CPU and memory)
    - Identifying wait time patterns, etc
- Query Store is not enabled by default for Synapse SQL but can be enabled with the below query
- Query Store stores the following main components in the associated system tables
    - Query details: Query parameters, compilation times, compilation counts, etc
        - sys.query_store_query
        - sys.query_store_query_text
    - Plan details: Query identifier (ID), query hash, query text, etc
        - sys.query_store_plan
    - Runtime statistics: Time taken, CPU time, average duration, row counts, etc
        - sys.query_store_runtime_stats
        - sys.query_store_runtime_stats_interval
    - Wait statistics (not available for Synapse SQL atm)

Spark Query performance Monitoring

- Spark performance monitoring can be done directly via the Spark User Interface (UI)
- Spark UI and Spark History Server provide job-specific and stage-specific metrics
    - Used to determine job performance relative to prior runs
    - Query optimization techniques can be used once slow queries are identified
- Covered in Ch 9, Section Debugging Spark Jobs Using Spark UI

Interpreting a Spark DAG

- A DAG is a regular graph with nodes and edges, but with no cycles or loops
- Spark DAGs can be accessed from the Spark UI by cicking a job link and then the DAG Visualization link

- When a user submits a Spark job, the Spark driver first identifies all the tasks involved in accomplishing the job. It then figures out which of these tasks can be run in parallel and which tasks depend on other tasks. Based on this information, it converts the Spark job into a graph of tasks. The nodes at the same level indicate jobs that can be run in parallel, and the nodes at different levels indicate tasks that need to be run after the previous nodes. This graph is acyclic, as denoted by A in DAG. This DAG is then converted into a physical execution plan. In the physical execution plan, nodes that are at the same level are segregated into stages. Once all the tasks and stages are complete, the Spark job is termed as completed.
- Learning to read Spark DAGs help identify bottlenecks in Spark queries

Monitoring Cluster Performance (Pg 427)

- Since Synapse and ADF are PaaS, we won't have explicit control over clusters
- The only service we can control clusters in is HDInsight
- The HDInsight portal has four main areas to monitor cluster performance in the Ambari Dashboard
    - Home: Monitor overall cluster performance and health
    - Hosts: Monitor per-node performance
    - Resource Manager: Yet Another Resource Negotiator (YARM) queries

Monitoring Storage Throttling

- Persistent [429: Too many requests] errors indicate storage throttling
- Check storage account to increase limit or try to distribute application processing over multiple storage accounts

Scheduling and Monitoring Pipeline Tests

- Azure DevOps provides a feature called Azure Pipelines
    - Used to create CI/CD pipelines to deploy ADF (not available for Synapse Pipelines)
    - Continuous Integration, Continuous Deployment
    - Method to automate testing and deployment to production environments

Creating a CI/CD Pipeline Using Azure Pipelines (Pg 431)

- In the Azure DevOps resource
  - All Pipelines > New Release Pipelines > New Pipeline
  - Select a template or start with an empty job
  - Add an artifact
  - Add ARM Template Deployment task
  - Configure Azure Resource Manager (ARM)
- Once completed, CI/CD pipelines can be scheduled multiple ways
  - CI Triggers: Triggered when a user pushes a change to the git branch
  - PR Triggers: Triggered when a Pull Request (PR) is raised or changes are submitted to a PR
  - Scheduled Triggers: Triggerd on a time-based scheduling
  - Pipeline Completion Triggers: Triggered when a previous pipeline is completed
- Pipelines will continue to get tested and deployed once configured with triggers
- Progress can be monitored from the Release tab in Azure DevOps

Summary

This chapter introduced a lot of new technologies and techniques, and I hope you got a grasp of them. Even though the number of technologies involved is high, the weightage of this chapter with respect to the certification is relatively low, so you may have noticed that I've kept the topics slightly at a higher level and have provided further links for you to read more on the topics.

In this chapter, we started by introducing Azure Monitor and Log Analytics. We learned how to send log data to Log Analytics, how to define custom logging options, and how to interpret metrics and logs data. After that, we focused on measuring the performance of data movements, pipelines, SQL queries, and Spark queries. We also learned how to interpret Spark DAGs, before moving on to monitoring cluster performance and cluster pipeline tests. You should now be able to set up a monitoring solution for your data pipelines and be able to tell if your data movement, pipeline setups, cluster setups, and query runs are performing optimally.

In the next chapter, we will be focusing on query optimization and troubleshooting techniques.

Chapter 14: Optimizing and Troubleshooting Data Storage and Data Processing

Compacting Small Files

- Analytical engines and cloud storage systems are optimized for big files
- Merging or compacting small files into bigger files makes for more efficient data pipelines

Compacting Small Files in ADF or Synapse Pipelines

- Select the Copy Data activity
- In the Source settings, select Wildcard File Path
  - Pass the source data file path with a * at the end to consider all files in the directory
  - staging / driver / in / *
- In the Sink settings, set the Copy Behavior to Merge Files
- All files in the directory will be merged to the output file

Compacting Small Files Using Incremental Copy

- Reviewed in Ch 4, Section: Desiging for Incremental Loading
- Tables are incrementally updated with small incoming changes

Compacting Small Files with Azure Databricks Spark and Synapse Spark

- Spark provides a feature called bin packing with Delta Lake tables
- Compacts small files into bigger ones to improve read query performance
- Enabled on a folder using the OPTIMIZE command
- Set to optimize by default when creating tables using auto optimize properties

```
# Enable Optimize on a Folder
Spark.sql("OPTIMIZE delta.' abfss://path/to/delta/files'")
```

```
# Optimize by Default
CREATE TABLE Customer (
    id INT
    , name STRING
    , location STRING
)
TBLPROPERTIES (
    delta.autoOptimize.optimizeWrite = True
    , delta.autoOptimize.optimizeCompact = True
);
```

Delta Lake Tables

- Open-source storage layer that runs on top of Data Lakes
- Enhances Data Lakes to support the following features
    - ACID Transactions
    - Unified Batch, Interactive, and Streaming Systems
    - Updates and Deletes
        - Enables automatic support for Slowly Changing Dimensions (SCDs)
- The Delta Lake engine is enabled by default in Azure Databricks and Synapse Spark
- Specified Azure Storage protocol
    - abfss: Azure Blob File System [Secure]

```
# Write Delta
df.write.mode("Overwrite").format("delta).save("abfss://path/to/delta/files")

# Load Delta
Val df: DataFrame = spark.read.format("delta").load("abfss://path/to/delta/files")

# Create Delta
Spark.sql("""
    CREATE TABLE CUSTOMER
    USING DELTA
    LOCATION "abfss://path/to/delta/files"
""")
```

Rewriting User-Defined Functions (UDFs)

- UDFs are custom functions that can be defined in databases and certain analytical /streaming engines (Spark, ASA)

Writing UDFs in Synapse SQL Pool

- Create UDFs in Synapse SQL using CREATE FUNCTION command
- Once created, a UDF can be called in the SELECT clause of a query passing any necessary parameters

Writing UDFs in Spark

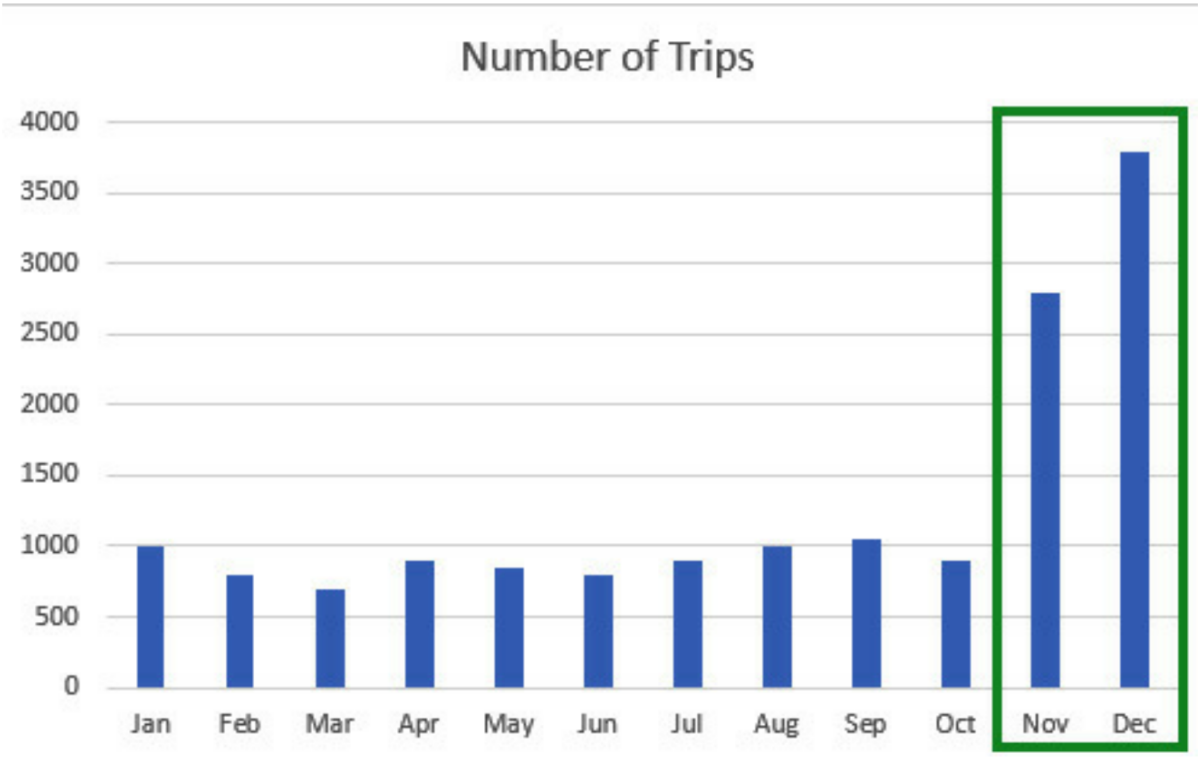
- UDFs in Spark are written like regular functiond
- UDFs can be used to reduce code duplication

Writing UDFs in Stream Analytics (ASA)

- ASA supports simple UDFs in JavaScript
- UDFs are stateless and can only return a single scalar value
- UDFs have to be registered in the portal
    - ASA Job page
    - Select Functions, + Add
    - Enter code in the snippet box
    - Once registered, a UDF can be called in the SELECT clause of a query passing any necessary parameters

Handling Skews in Data

- Data skew refers to extreme, uneven distribution of data in a dataset
- Data skew can degrade performance as certain nodes will have to process more data
    - Example: In the screenshot below, the data is distributed by month so the nodes processing Nov and Dec data will take more time and bottleneck the other nodes
- Data skew results in inefficient job performances
- To make processing more efficient, will have to find a way to assign similar amounts of processing data to each compute node
    - Skews can be fixed at the storage level or the compute level



Fixing Skews at the Storage Level

- Find a better distribution/partition strategy that would balance compute evenly
- If a better distribution/partition strategy is not possible, add a second distribution key
- Last case, randomize the data and use the round-robin technique to distribute data evenly

Fixing Skews at the Compute Level

- Improve query performance by enabling statistics
  - Query engines that use cost-based optimizer will use statistics to generate the most optiomal query plan
- Ignore outlier data if not significant

Identifying Skews at the Compute Level

- Synapse Spark has a data skew feature
- Monitoring > Apache Spark Applications > View skew details

Handling Data Spills

- When a compute engine executing a query is unable to hold the required data in memory and writes (spills) to disk
- Results in increased query execution time due to expensive disk reads and writes
- Spills can occur for the following reasons
    - Data partition is too big
    - Compute resource size is small (especially memory)
    - Exploded data size during merges, unions, etc
        - Exceeding memorylimits of the compute node
- Solutions for handling data spills
    - Increase the compute capacity
    - Reduce partition sizes and repartition
    - Remove skews in data
        - Skews can cause spills in the partitions with more data being processed

Identifying Data Spills in Synapse SQL

- Primary indicator for data spills is TempDB errors (running out of space)
- - Azure provides queries to monitor
    - Memory usage
    - TempDB usage

Identifying Data Spills in Spark

- Spark publishes metrics for spills
- Check Tasks Summary screen from Spark UI
- Column: Spill (Disk) indicates the bytes of data written to disk (spilled)

Tuning Shuffle Partitions

- Spark uses a technique called Shuffle to move data between executor nodes
    - While performing operations such as join, union, groupby, reduceby
- Shuffle is a very expensive operation, preferable to reduce amount of shuffle
- Number of partition splits that Spark performs while Shuffling data is determined by the below configuration
    - 200 is the default, tune to best suit the query

```
spark.conf.set("spark.sql.shuffle.partitions", 200)
```

Finding Shuffling in a Pipeline

Identifying Shuffles in a SQL Query Plan

- Identify shuffles by printing the query plan using the EXPLAIN WITH_RECOMMENDATIONS statement
- Query will print an XML plan
- Look for keyword SHUFFLE_MOVE to identify
  - Shuffle stages
  - Cost details
  - Rows invovled
  - Query causing shuffle, etc
- Read query plans bottom up

Identifying Shuffles in Spark Query Plan

- Similar to SQL, we can use the explain command to print the plans in Spark
  - df.explain
- Look for keyword EXCHANGE to identify shuffle stages
- Alternatively, identify shuffle stage from a Spark DAG

Optimizing Resource Management (Billing Expenses)

Optimizing Synapse SQL Pools

- Pause SQL pool compute when not in use
- Use the right size of compute units (DWUs)
- Manually scale compute resources out or in based on load
  - Can sutomate with Azure Functions

Optimizing Spark

- Select auto-scale option when setting up cluster
- Select auto-terminate option to terminate cluster after the configured time

- Choose spot-instances where available
  - Cheap nodes with no availability guarantee
- Choose the right type of cluster nodes based on memory-intensive, CPU-intensive, or network-intensive jobs
  - Always select nodes that have more memory than the maximum memory required by the job

Tuning Queries by Using Indexes

Indexing in Synapse SQL

- Reviewed in Ch 5, Section: Implementing Different Table Geometries with Azure Synapse Analytics Pools
- Three primary indexing available in Synpase SQL
  - Clustered Columnstore Index
    - Default index
    - Used for large tables (> 100m rows)
    - Provides high levels of data compression and good overall performance
  - Clustered Index
    - Used for smaller tables (< 100m rows)
    - Used for specific filter conditions
  - Heap Index
    - Used for temporary staging tables
    - Used for small lookup tables and tables with transient data
- If index performance degrades over time, try rebuilding indexes
- Secondary non-clustered index can be built on top of a clustered index to speed up filtering

Indexing in Synapse Spark Pool Using Hyperspace

- Spark does not support builtin indexing options
- Microsoft Hyperspace can be used to create indexes in Spark
- Hyperspace supports common file formats (CSV, JSON, Parquet)
- Hyperspace provides a simple set of APIs that can be used to create ana manage indexes

Tuning Queries by Using Cache

- Caching is a well-known method for improving read performance in databases
- Synapse SQL supports a feature called Result Set Caching
  - Enables results to be cached and reused if the query doesn't change
- Result set cache is only used under the following conditions
  - Query being considered is an exact match
  - There are no changes to the underlying data or schema
  - The user has the right set of permissions to the tables references in the query
- Can be enabled at the database level or within a session
- Maximum result set cache is 1TB per database
  - Old data is evicted when maximum size is reached

```
-- Database Level
ALTER DATABASE [database_name]
SET RESULT_SET_CACHING ON;

-- Session Level
SET RESULT_SET_CACHING ON | OFF;
```

Caching in Spark

- Also supported, much smaller scope
- Functions cache() and persist() can be used to cache intermediate results of RDDs, DataFrames, or datasets

Optimizing Pipelines for Analytical or Transactional Purposes

OLTP Systems

- Online Transactional Processing Systems
- Built to efficiently process, store, and query transactions
- Transaction data flows into a central ACID compliant database
- Databases are normalized and adhere to strict schemas
- Predominantly RDBMS-based systems
  - Azure SQL, MySQL

OLAP Systems

- Online Analytical Processing
- Big data systems with a warehouse or key-value based store
- Data arrives from various sources so strict schemas or formats are not typically enforced
- Typically large amounts of data in column-based formats
    - Synapse SQL, Pool, HBase, CosmosDB

Optimizing Pipelines for Descriptive vs Analytical Workloads

- Data is categorized in four different categories
    - Descriptive Analytics
    - Diagnostic Analytics
    - Predictive Analytics
    - Prescriptive Analytics

Common Optimizations for Descriptive and Analytical Pipelines

- Common techniques irrespective of SQL Pool centric (descriptive) or Spark Centric (analytical) approaches
- Optimizations at the storage level
    - Divide data into zones: Raw, Transformation, and Serving zones
    - Define a good directory structure, organized around dates
    - Partition data based on access to different directories and different storage tiers
    - Choose the right data storage format (CSV, Parquet, etc)
    - Configure the data lifecycle, purging old data or moving it to archive tiers
- Optimizations at the compute level
    - Use caching
    - Use indexing when available
    - Handle data spills
    - Handle data skews
    - Tune queries by reading query plans

Specific Optimizations for Descriptive and Analytical Pipelines

- Synapse SQL optimizations
  - Maintain statistic to improve performance while using cost-based optimizer
  - Use PolyBase to load data faster
  - Use hash distribution for large tables
  - Use temporary heap tables for transient data
  - Do not over partition (Synapse SQL already partitions into 60)
  - Minimize transaction sizes
  - Reduce query result sizes
  - Use Result Set Cache is necessary
  - Use the smallest possible column size when creating tables
  - Use a larger resource class to increasequery performance (larger memory size)
  - Use a smaller resource class to increase concurrency (smaller memory size)
- Spark optimizations
  - Choose the right data abstraction: DataFrames and datasets usually work better than RDDs
  - Choose the right data format: Parquet with a Snappy compression usually works fine for the majority of Spark use cases
  - Use cache: Either the inbuilt ones in Spark, such as .cache() and .persist(), or external caching libraries
  - Use indexers: Use Hyperspace to speed up queries
  - Tune your queries: Reduce shuffles in the query plan, choose the right kind of merges, and so on
  - Optimize job execution: Choosing the right container sizes so that the jobs don't run out of memory
    - This can usually be done by observing the logs for the details of previous runs.

Troubleshooting a Failed Spark Job

- Two aspects to troubleshooting a failed Spark job
  - Environmental issues (service)
    - Check health of Azure services in specified region
    - Check health of Spark cluster

        - Check metrics such as high CPU or memory usage
    - Job issues
        - Check log files
            - Driver Logs
            - Tasks Logs
            - Executor Logs

Troubleshooting a Failed Pipeline Run

- ADF and Synpase Pipeline rovide detailed error messages when pipelines fail
- Common troubleshooting steps
    - Check Linked Services (connections) for datasets
    - Use data preview to check transformations (Data Flow Debug must be on)
    - Check error message detailed on failed pipeline runs

Summary

Like the last chapter, this chapter also introduced a lot of new concepts. Some of these concepts will take a long time to master, such as Spark debugging, optimizing shuffle partitions, and identifying and reducing data spills. These topics could be separate books on their own. I've tried my best to give you an overview of these topics with follow-up links. Please go through the links to learn more about them.

Let's recap what we learned in this chapter. We started with data compaction as small files are very inefficient in big data analytics. We then learned about UDFs, and how to handle data skews and data spills in both SQL and Spark. We then explored shuffle partitions in Spark. We learned about using indexers and cache to speed up our query performance. We also learned about HTAP, which was a new concept that merges OLAP and OLTP processing. We then explored the general resource management tips for descriptive and analytical platforms. And finally, we wrapped things up by looking at the guidelines for debugging Spark jobs and pipeline failures.

You should now know the different optimizations and query tuning techniques. This should help you with both certification and becoming a good Azure data engineer.

You have now completed all the topics listed in the syllabus for DP-203 certification. Congratulations to you for your persistence in reaching the end!

The next chapter will cover sample questions to help you prepare for the exam.
