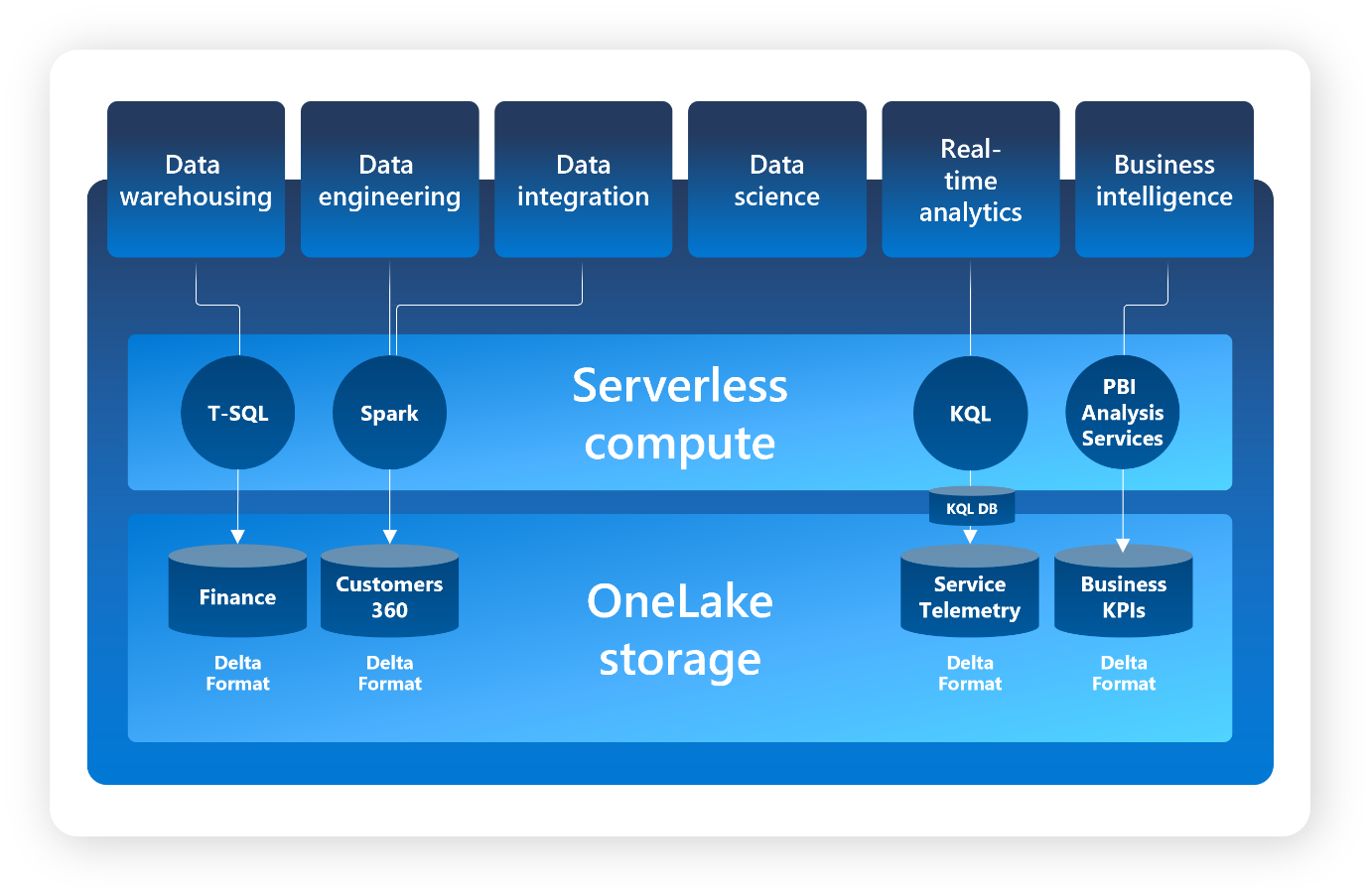
**AI Skills Challenge: Fabric Analytics Engineer**

prepare for Exam DP-600

[Microsoft Fabric documentation - Microsoft Fabric | Microsoft Learn](https://learn.microsoft.com/en-us/fabric/)

[Welcome to the Delta Lake documentation — Delta Lake Documentation](https://docs.delta.io/latest/index.html)

## Introduction to end-to-end analytics using Microsoft Fabric

* Fabric offers persona-optimized experiences and tools in an integrated user interface.
* Fabric is a unified software-as-a-service (SaaS) offering, with all your data stored in a single open format in OneLake.
* OneLake is Fabric's lake-centric architecture that provides a single, integrated environment for data professionals and the business to collaborate on data projects.
* Fabric's OneLake architecture facilitates collaboration between data team members and saves time by eliminating the need to move and copy data between different systems and teams.
* OneCopy is a key component of OneLake that allows you to read data from a single copy, without moving or duplicating data.
* OneLake combines storage locations across different regions and clouds into a single logical lake, without moving or duplicating data.
* all the compute workloads in Fabric are preconfigured to work with OneLake. Fabric's data warehousing, data engineering (Lakehouses and Notebooks), data integration (pipelines and dataflows), real-time analytics, and Power BI all use OneLake as their native store without needing any extra configuration.
* OneLake is built on top of Azure Data Lake Storage (ADLS) and data can be stored in any format, including Delta, Parquet, CSV, JSON, and more.
* or tabular data, the analytical engines in Fabric will write data in delta-parquet format and all engines interact with the format seamlessly.
* One important feature of OneLake is the ability to create shortcuts, which are embedded references within OneLake that point to other files or storage locations.
* Fabric administration is centralized in the admin center. In the admin center you can manage groups and permissions, configure data sources and gateways, and monitor usage and performance. For more information about Fabric administration, see [What is Microsoft Fabric admin](https://learn.microsoft.com/en-us/fabric/admin/microsoft-fabric-admin).
* Your Fabric tenant is natively integrated with Microsoft Purview Information Protection. Fabric uses Microsoft Purview Information Protection’s sensitivity labels to help your organization classify and protect sensitive data, from ingestion to export.
* If you have admin privileges, you can access the **Admin center** from the **Settings** menu in the upper right corner of the Power BI service. From here, you enable Fabric in the **Tenant settings.**
* Admins can make Fabric available to either the entire organization or specific groups of users, who can be organized based on their Microsoft 365 or Microsoft Entra security groups. Admins can also *delegate* the ability to enable Fabric to other users, at the capacity level.
* All Fabric items (lakehouses, notebooks, pipelines, etc.) are stored in OneLake and accessed via Fabric workspaces. Workspaces must be in Premium capacity to use Fabric. If you don't have access to Premium capacity, you aren't able to use Fabric. Select **Trial** in the **Premium capacity settings** section of the **Workspace settings** page to enable Premium capacity for your workspace.
* 

## Get started with lakehouses in Microsoft Fabric

* The foundation of Microsoft Fabric is a lakehouse, which is built on top of the OneLake scalable storage layer and uses Apache Spark and SQL compute engines for big data processing.
* A lakehouse is a unified platform that combines:
  + The flexible and scalable storage of a data lake
  + The ability to query and analyze data of a data warehouse.
* A **lakehouse** presents as a database and is built on top of a data lake using Delta format tables. Lakehouses combine the SQL-based analytical capabilities of a relational data warehouse and the flexibility and scalability of a data lake.
* **Lakehouse = Date Lake + Data Warehouse**
* As cloud-based solutions, lakehouses can scale automatically and provide high availability and disaster recovery.
* Lakehouse benefits:
  + uses Spark and SQL engines.
  + Lakehouse data is organized in a schema-on-read format, which means you define the schema as needed rather than having a predefined schema.
  + Lakehouses support ACID (Atomicity, Consistency, Isolation, Durability) transactions through Delta Lake formatted tables.
  + Lakehouses are a single location for data engineers, data scientists, and data analysts to access and use data.
* We can work with the data in the lakehouse in two modes:
  + Lakehouse enables you to add and interact with tables, files, and folders in the lakehouse.
  + SQL analytics endpoint enables you to use SQL to query the tables in the lakehouse and manage its relational data model.
* **Shortcuts** enable you to integrate data into your lakehouse while keeping it stored in external storage.
* Shortcuts can be created in both lakehouses and KQL databases, and appear as a folder in the lake. Spark, SQL, Real-Time Analytics, and Analysis Services can access data via shortcuts when querying data. [OneLake shortcuts - Microsoft Fabric | Microsoft Learn](https://learn.microsoft.com/en-us/fabric/onelake/onelake-shortcuts)
* Ways to load data into a Fabric lakehouse, including:
  + Upload
  + Dataflows (Gen2): Import and transform data from a range of sources using Power Query Online, and load it directly into a table in the lakehouse.
  + Notebooks: Use notebooks in Fabric to ingest and transform data, and load it into tables or files in the lakehouse.
  + Data Factory pipelines: Copy data and orchestrate data processing activities, loading the results into tables or files in the lakehouse.
* Explore and transform data in a lakehouse, including:
  + Apache Spark: use Spark pools through Notebooks or Spark Job Definitions to process data in files and tables in the lakehouse using Scala, PySpark, or Spark SQL.
  + SQL analytic endpoint
  + Dataflows (Gen2)
  + Data pipelines: operates on data in the lakehouse through a sequence of activities (such as dataflows, Spark jobs, and other control flow logic).
  + Exercise Create a Lakehouse: [mslearn-fabric (microsoftlearning.github.io)](https://microsoftlearning.github.io/mslearn-fabric/Instructions/Labs/01-lakehouse.html)

## Use Apache Spark in Microsoft Fabric

* Spark uses a data structure called a resilient distributed dataset (RDD).
* data structure for working with structured data in Spark is the dataframe, which is provided as part of the Spark SQL library.
* Loading data into a dataframe:
  + Python

|  |
| --- |
| %%pyspark  df = spark.read.load('Files/data/products.csv',  format='csv',  header=True  )  display(df.limit(10)) |

* + Scala

|  |
| --- |
| %%spark  val df = spark.read.format("csv").option("header", "true").load("Files/data/products.csv")  display(df.limit(10)) |

* Specifying an explicit schema

|  |
| --- |
| from pyspark.sql.types import \*  from pyspark.sql.functions import \*  productSchema = StructType([  StructField("ProductID", IntegerType()),  StructField("ProductName", StringType()),  StructField("Category", StringType()),  StructField("ListPrice", FloatType())  ])  df = spark.read.load('Files/data/product-data.csv',  format='csv',  schema=productSchema,  header=False)  display(df.limit(10)) |

* + Specifying an explicit schema also improves performance!
* Filtering and grouping dataframes
  + pricelist\_df = df.select("ProductID", "ListPrice")
  + Or
  + pricelist\_df = df["ProductID", "ListPrice"]
  + Chain, Group by:

|  |
| --- |
| bikes\_df = df.select("ProductName", "Category", "ListPrice").where((df["Category"]=="Mountain Bikes") | (df["Category"]=="Road Bikes"))  display(bikes\_df)  counts\_df = df.select("ProductID", "Category").groupBy("Category").count()  display(counts\_df) |

* Saving a dataframe:

|  |
| --- |
| bikes\_df.write.mode("overwrite").parquet('Files/product\_data/bikes.parquet') |

* Partitioning the output file
  + Partitioning is an optimization technique that enables Spark to maximize performance across the worker nodes. More performance gains can be achieved when filtering data in queries by eliminating unnecessary disk IO.
  + Example to save a dataframe as a partitioned set of files:
  + bikes\_df.write.partitionBy("Category").mode("overwrite").parquet("Files/bike\_data")
* Load Partition Data

|  |
| --- |
| road\_bikes\_df = spark.read.parquet('Files/bike\_data/Category=Road Bikes')  display(road\_bikes\_df.limit(5)) |

* Work with data using Spark SQL
  + The Dataframe API is part of a Spark library named Spark SQL, which enables data analysts to use SQL expressions to query and manipulate data.
  + The Spark catalog is a metastore for relational data objects such as views and tables.
  + One of the simplest ways to make data in a dataframe available for querying in the Spark catalog is to create a temporary view, A view is temporary, meaning that it's automatically deleted at the end of the current session.
    - df.createOrReplaceTempView("products\_view")
  + Tables are metadata structures that store their underlying data in the storage location associated with the catalog.
  + In Microsoft Fabric, data for managed tables is stored in the **Tables** storage location shown in your data lake, and any tables created using Spark are listed there.
  + You can create an empty table by using the spark.catalog.createTable method, or you can save a dataframe as a table by using its saveAsTable method
    - df.write.format("delta").saveAsTable("products")
  + Deleting a managed table also deletes its underlying data.
  + The Spark catalog supports tables based on files in various formats. The preferred format in Microsoft Fabric is **delta**, which is the format for a relational data technology on Spark named Delta Lake. Delta tables support features commonly found in relational database systems, including transactions, versioning, and support for streaming data.
  + Additionally, you can create external tables by using the spark.catalog.createExternalTable method. External tables define metadata in the catalog but get their underlying data from an external storage location; typically a folder in the **Files** storage area of a lakehouse. Deleting an external table doesn't delete the underlying data.
  + You can apply the same partitioning technique to delta lake tables as discussed for parquet files in the previous unit. Partitioning tables can result in better performance when querying them.
  + Using the Spark SQL API to query data
    - Python

|  |
| --- |
| bikes\_df = spark.sql("SELECT ProductID, ProductName, ListPrice \  FROM products \  WHERE Category IN ('Mountain Bikes', 'Road Bikes')")  display(bikes\_df) |

* + Using SQL code

|  |
| --- |
| %%sql  SELECT Category, COUNT(ProductID) AS ProductCount  FROM products  GROUP BY Category  ORDER BY Category |

## Work with Delta Lake tables in Microsoft Fabric

* [Work with Delta Lake tables in Microsoft Fabric - Training | Microsoft Learn](https://learn.microsoft.com/en-us/training/modules/work-delta-lake-tables-fabric/)
* Use delta tables with streaming data. Spark Structured Streaming
  + A typical stream processing solution involves constantly reading a stream of data from a source, optionally processing it to select specific fields, aggregate and group values, or otherwise manipulate the data, and writing the results to a sink.
  + Spark includes native support for streaming data through **Spark Structured Streaming**, an API that is based on a boundless dataframe in which streaming data is captured for processing. A Spark Structured Streaming dataframe can read data from many different kinds of streaming source, including network ports, real time message brokering services such as Azure Event Hubs or Kafka, or file system locations.
  + [Structured Streaming Programming Guide - Spark 3.5.1 Documentation (apache.org)](https://spark.apache.org/docs/latest/structured-streaming-programming-guide.html)
  + **The key idea in Structured Streaming is to treat a live data stream as a table that is being continuously appended**
* Streaming with delta tables
  + You can use a delta table as a source or a sink for Spark Structured Streaming.
  + Using a delta table as a streaming source.In the following PySpark example, a delta table is used to store details of Internet sales orders. A stream is created that reads data from the table folder as new data is appended.

|  |
| --- |
| from pyspark.sql.types import \*  from pyspark.sql.functions import \*  # Load a streaming dataframe from the Delta Table  stream\_df = spark.readStream.format("delta") \  .option("ignoreChanges", "true") \  .load("Files/delta/internetorders")  # Now you can process the streaming data in the dataframe  # for example, show it:  stream\_df.show() |

* + Note: When using a delta table as a streaming source, only append operations can be included in the stream. Data modifications will cause an error unless you specify the ignoreChanges or ignoreDeletes option.
* Using a delta table as a streaming sink
  + In the following PySpark example, a stream of data is read from JSON files in a folder. The JSON data in each file contains the status for an IoT device in the format {"device":"Dev1","status":"ok"} New data is added to the stream whenever a file is added to the folder. The input stream is a boundless dataframe, which is then written in delta format to a folder location for a delta table.

|  |
| --- |
| from pyspark.sql.types import \*  from pyspark.sql.functions import \*  # Create a stream that reads JSON data from a folder  inputPath = 'Files/streamingdata/'  jsonSchema = StructType([  StructField("device", StringType(), False),  StructField("status", StringType(), False)  ])  stream\_df = spark.readStream.schema(jsonSchema).option("maxFilesPerTrigger", 1).json(inputPath)  # Write the stream to a delta table  table\_path = 'Files/delta/devicetable'  checkpoint\_path = 'Files/delta/checkpoint'  delta\_stream = stream\_df.writeStream.format("delta").option("checkpointLocation", checkpoint\_path).start(table\_path) |

* + Note:The checkpointLocation option is used to write a checkpoint file that tracks the state of the stream processing. This file enables you to recover from failure at the point where stream processing left off.
  + After the streaming process has started, you can query the Delta Lake table to which the streaming output is being written to see the latest data. For example, the following code creates a catalog table for the Delta Lake table folder and queries it:

|  |
| --- |
| %%sql  CREATE TABLE DeviceTable  USING DELTA  LOCATION 'Files/delta/devicetable';  SELECT device, status  FROM DeviceTable; |

* + To stop the stream of data being written to the Delta Lake table, you can use the stop method of the streaming query:
    - delta\_stream.stop()
* Tip: For more information about using delta tables for streaming data, see [Table streaming reads and writes — Delta Lake Documentation](https://docs.delta.io/latest/delta-streaming.html).
* Exercise: [mslearn-fabric (microsoftlearning.github.io)](https://microsoftlearning.github.io/mslearn-fabric/Instructions/Labs/03-delta-lake.html)

## Knowledge check

* Which of the following descriptions best fits Delta Lake? A relational storage layer for Spark that supports tables based on Parquet files.
* You've loaded a Spark dataframe with data, that you now want to use in a delta table. What format should you use to write the dataframe to storage? DELTA
* You have a managed table based on a folder that contains data files in delta format. If you drop the table, what happens? The table metadata and data files are deleted.