**Madiha Aimon Tappal**

[madihaaimon@gmail.com](mailto:madihaaimon@gmail.com)

**Data Engineering Batch – 1**

**Coding Challenge**

**Azure Databricks**

1. **Exploratory data analysis (EDA) in Databricks &Visualizing data in Databricks**

**What is EDA?**

EDA involves investigating and understanding the underlying patterns and characteristics of a dataset. You use statistical methods and data visualizations to uncover trends, identify anomalies, and assess data quality before diving into further analysis or building models.

**What is EDA and why is it useful?**

Exploratory data analysis (EDA) includes methods for exploring data sets to summarize their main characteristics and identify any problems with the data. Using statistical methods and visualizations, you can learn about a data set to determine its readiness for analysis and inform what techniques to apply for data preparation. EDA can also influence which algorithms you choose to apply for training ML models.

## What are the EDA tools in Databricks?

Databricks has built-in analysis and visualization tools in both Databricks SQL and in Databricks Runtime. For an illustrated list of the types of visualizations available in Databricks, see [Visualization types](https://docs.databricks.com/en/visualizations/visualization-types.html).

**Key steps for EDA in Databricks:**

1. **Data loading and understanding:** Load your data using Databricks APIs or SQL and explore its schema and basic statistics.
2. **Data cleaning and wrangling:** Identify and address missing values, outliers, and inconsistencies.
3. **Univariate analysis:** Analyze individual variables using techniques like histograms, boxplots, and frequency tables.
4. **Bivariate analysis:** Explore relationships between pairs of variables using scatter plots, correlation analysis, and heatmaps.
5. **Multivariate analysis:** Investigate relationships between multiple variables using techniques like Principal Component Analysis (PCA).

### **EDA in Databricks SQL**

Here are some helpful articles about data visualization and exploration tools in Databricks SQL:

* [Visualize queries and create a dashboard in Databricks SQL](https://docs.databricks.com/en/sql/get-started/visualize-data-tutorial.html)
* [Create data visualizations in Databricks SQL](https://docs.databricks.com/en/sql/user/visualizations/index.html)

**Architecture of EDA in Azure Databricks**

When talking about Azure Databricks, specifically, architecture for Exploratory Data Analysis (EDA) takes a slightly different perspective than event-driven architecture (EDA). While event-driven architectures focus on reacting to data changes, EDA in Databricks focuses on exploring and understanding existing data to gain insights.

Here's a breakdown of the architecture for EDA in Azure Databricks:

**Components:**

1. **Data Sources:** The starting point is your data, residing in various sources like databases, cloud storage (Azure Blob Storage, ADLS Gen2), data warehouses, or other platforms.
2. **Ingestion & Preprocessing:** Data is ingested into Databricks using Databricks Jobs, APIs, or notebooks. Preprocessing steps like cleaning, filtering, and transforming might be applied.
3. **Delta Lake:** As a central storage layer, Delta Lake tables efficiently store and organize the data with ACID transactions and versioning for reliable analysis.
4. **Databricks Notebooks:** Interactive notebooks serve as your main exploration environment. You can use libraries like Pandas, Spark MLlib, and visualization libraries like Matplotlib, Plotly, and Vega-Lite to perform analysis and create visualizations.
5. **Collaboration & Sharing:** Notebooks can be shared with other users for interactive exploration and knowledge sharing. Version control and commenting facilitate collaborative work.
6. **Dashboards & Reporting:** Insights gained from EDA can be translated into reports or dashboards using tools like Databricks SQL Visualizations, Power BI, or custom integrations.

# **Visualization in Databricks:-**

Databricks offers several robust features for data visualization, enabling you to explore and understand your data effectively. Here's a breakdown of the key aspects:

**Visualization types:**

* **Built-in visualizations:** Databricks notebooks provide built-in support for various visualization types through libraries like:
  + **Plotly:** Create interactive charts and plots for diverse data exploration.
  + **Matplotlib:** Generate standard static visualizations like line plots, histograms, and scatter plots.
  + **Vega-Lite:** Leverage a powerful grammar-based approach for creating custom visualizations.
* **Third-party libraries:** You can access and utilize libraries like seaborn, ggplot2, and Bokeh within Databricks notebooks for even more diverse visualization options.

**Creating visualizations:**

1. **Data source:** Start with your data, whether it's in a Delta Lake table, DataFrame, or query result.
2. **Visualization selection:** Choose the appropriate visualization type based on your data and analysis goals.
3. **Configuration:** Specify visualization parameters like chart type, axes, colors, legends, and other desired customizations.
4. **Interaction & Exploration:** Explore the visualization interactively to gain deeper insights. Some visualizations offer features like zooming, panning, filtering, and data point inspection.

**Sharing & Collaboration:**

* Save visualizations within notebooks for future reference and reuse.
* Share notebooks with others for collaborative exploration and analysis.
* Embed visualizations in Databricks dashboards for wider dissemination and communication of insights.

**Additional features:**

* **Dashboarding:** Create interactive dashboards using tools like Databricks SQL Visualizations or Power BI. These dashboards can combine diverse visualizations from different sources for comprehensive data representations.
* **MLflow integration:** Visualize model performance metrics and data distributions within MLflow for machine learning exploration.
* **Customization:** Use custom JavaScript or HTML widgets within notebooks to create highly tailored visualizations for specific scenarios.

**Benefits of using Databricks for visualization:**

* **Scalability:** Handle large datasets efficiently with optimized visualization libraries.
* **Integration:** Seamlessly work with your data stored in Databricks or other connected sources.
* **Collaboration:** Leverage collaborative notebooks and dashboards for shared insights.
* **Variety:** Choose from various built-in and third-party libraries for diverse visualization needs.

**Creating visualizations in Azure Databricks** involves using a combination of code, data manipulation, and visualization libraries. Here are the general steps to create visualizations in Azure Databricks:

**1. Set Up Azure Databricks:**

* Ensure you have an active Azure Databricks workspace.
* Create a cluster to run your code.

**2. Create or Import a Notebook:**

* Open the Azure Databricks workspace.
* Create a new notebook or import an existing one.

**3. Choose a Language:**

* Decide on the programming language for your notebook (e.g., Scala, Python, SQL, or R).

**4. Load Data:**

* Load your data into a DataFrame or another suitable data structure.
* You can read data from various sources, such as Azure Data Lake Storage, Azure Blob Storage, or external databases.

**5. Data Exploration:**

* Explore your data using Spark DataFrame operations or SQL queries in the notebook cells.
* Ensure you understand the structure and characteristics of your data.

**6. Data Cleaning and Transformation:**

* Preprocess your data as needed, handling missing values, transforming columns, or aggregating information.

**7. Choose a Visualization Library:**

* Select a visualization library based on your chosen programming language.
* Common libraries for Python include Matplotlib, Seaborn, and Plotly.
* For R, you might use ggplot2 or other relevant packages.

**8. Create Visualizations:**

* Write code to generate visualizations based on your data.
* Use functions provided by the chosen library to create charts, graphs, or other visual representations.
* Include the visualization code in separate notebook cells.

**9. Display Visualizations:**

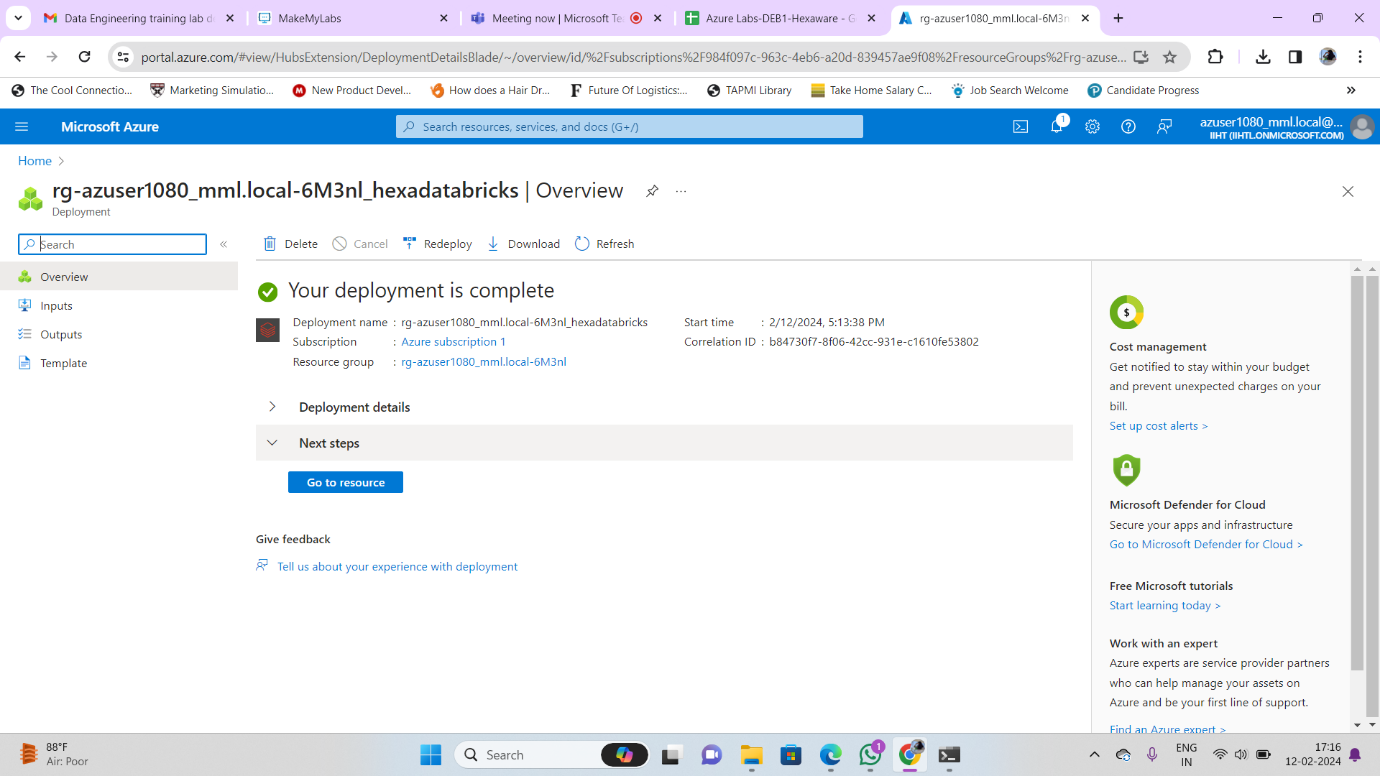
* Execute the cells containing visualization code to generate the charts.
* Visualizations are typically displayed directly below the code cells in the notebook.

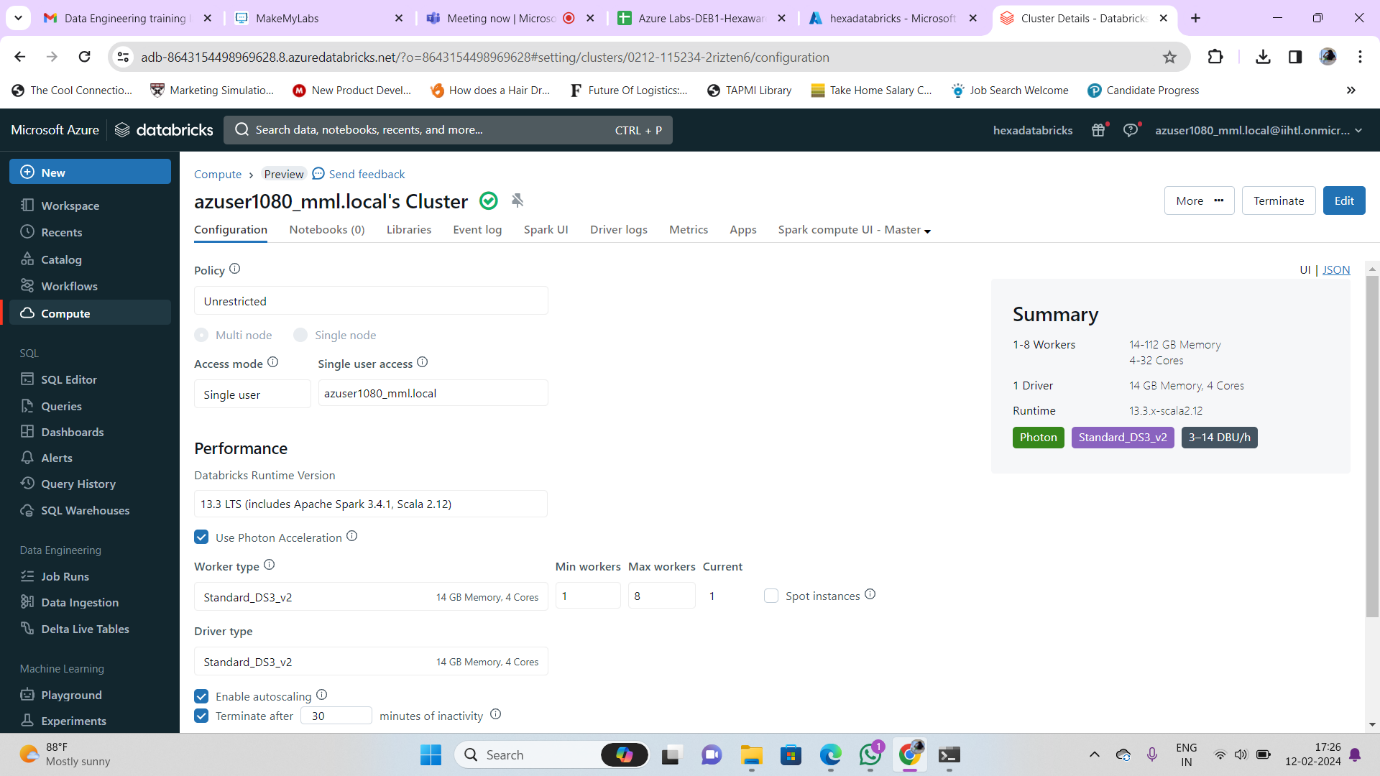
**Creating new visualizations:**

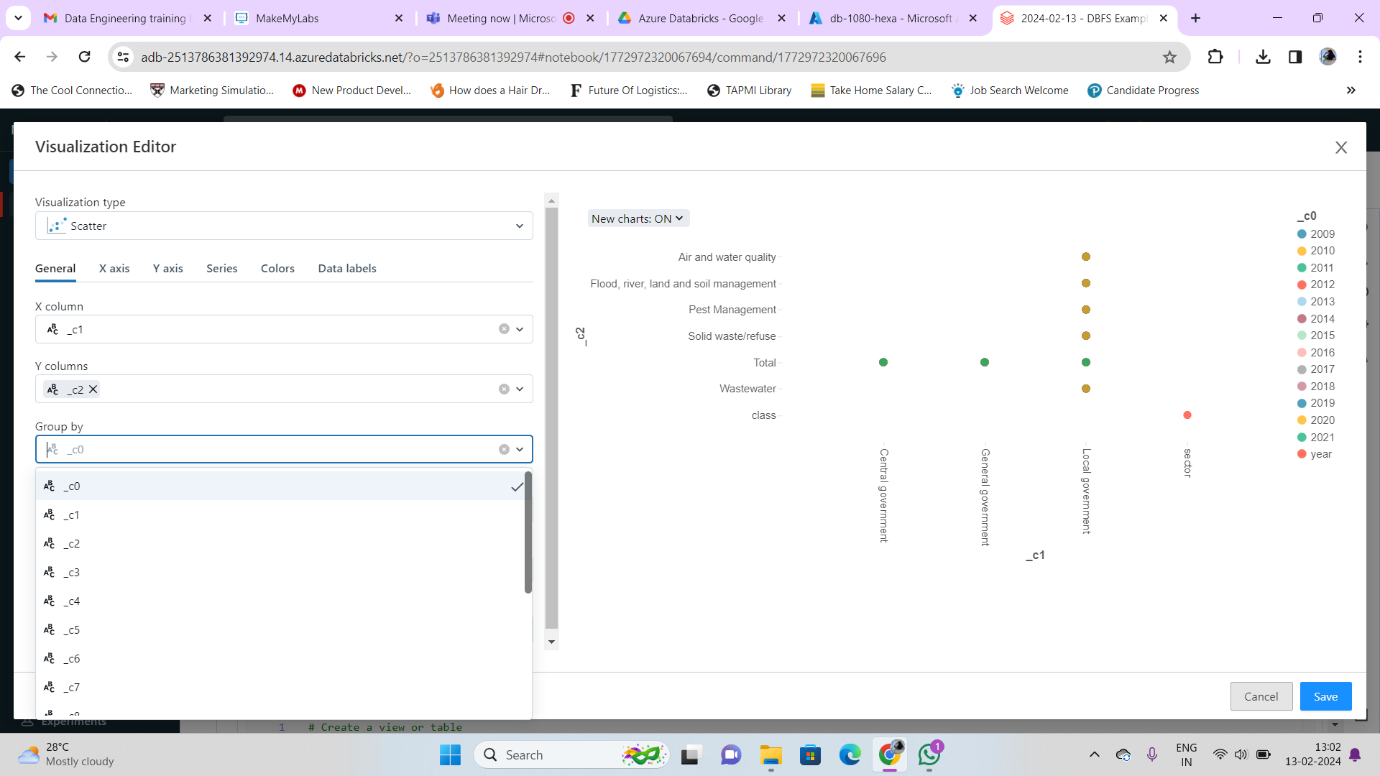
In order to create visualizations, we need to have data.

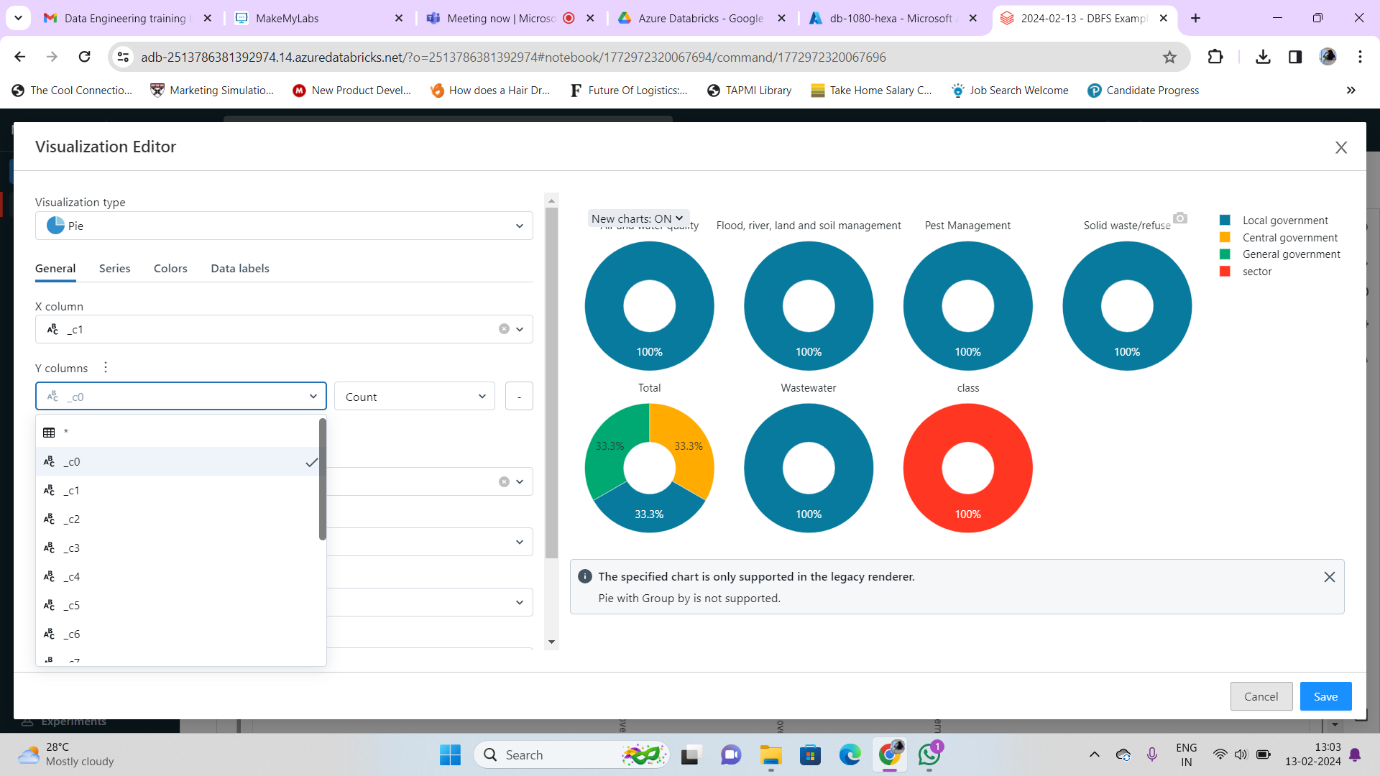
Here I provided a data and used display() method.

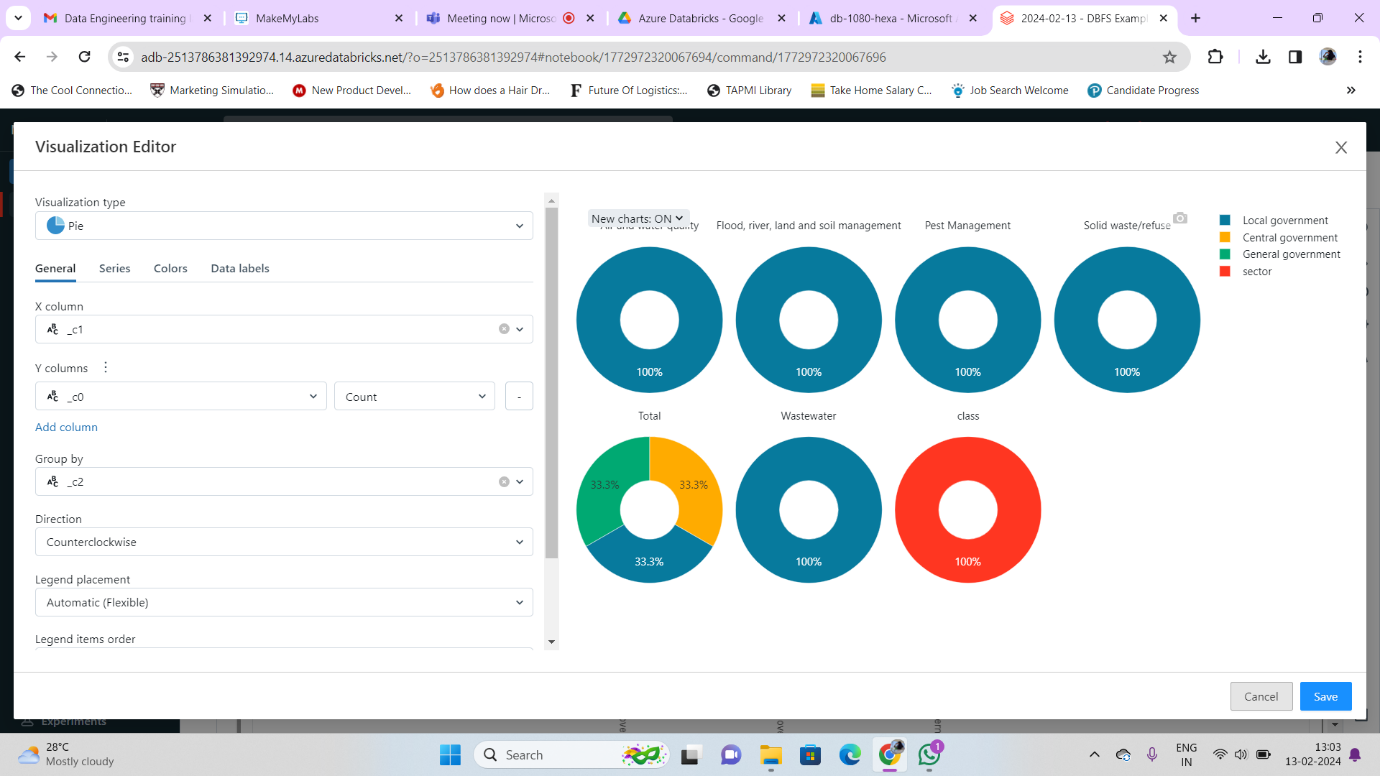
* After creating a table
* Click on + symbol
* Click on visualization.
* Select the type of visualization, here I selected “Scatter”

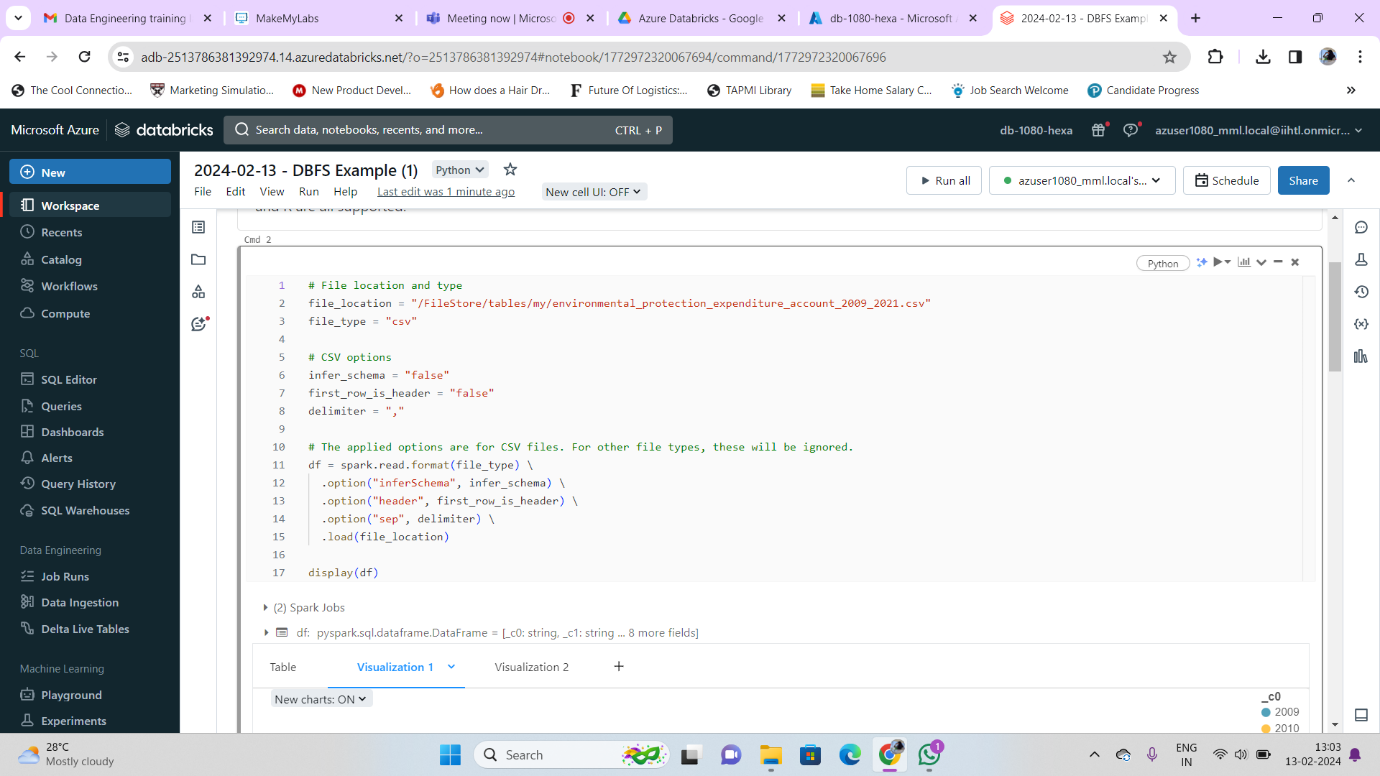


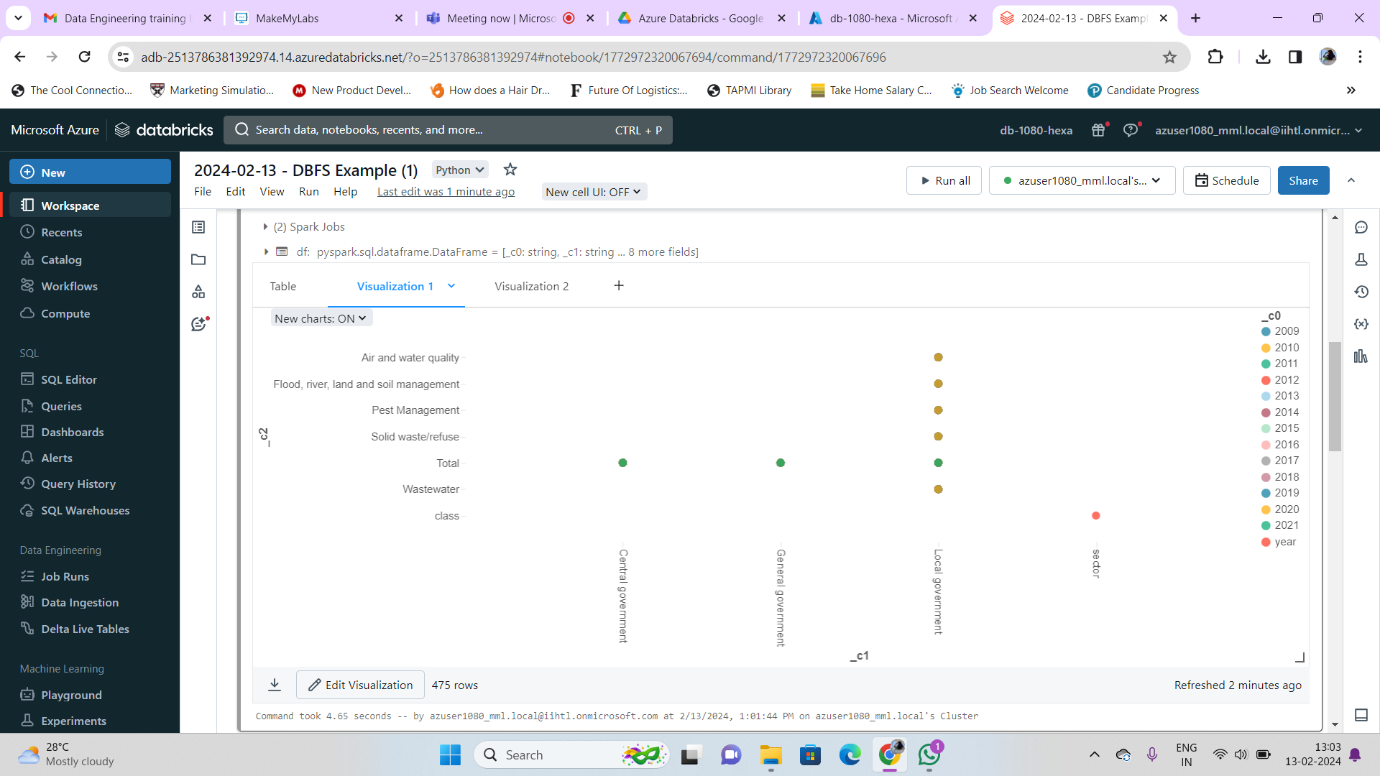












1. **Explain Overview of 3 level namespace and creating Unity Catalog objects.**

**Unity Catalog in Databricks:**

## Centralized Governance for Your Data

Unity Catalog is a powerful tool in Databricks that offers **centralized access control, auditing, lineage, and data discovery capabilities** across all your workspaces. Think of it as the single source of truth for your data governance needs.

**Key Features:**

* **Define Once, Secure Everywhere:** Set access policies for datasets and manage user permissions in one place, eliminating the need for repetitive configurations across workspaces.
* **Standards-Compliant Security:** Leverage familiar ANSI SQL syntax to grant permissions at various levels (catalogs, databases, tables, views), ensuring consistency and ease of use.
* **Automatic Data Lineage:** Track how data flows through your pipelines, from source to consumption, without manual setup. This transparency empowers data quality management and impact analysis.
* **Centralized Auditing:** Get a comprehensive view of all data access and usage across your workspaces, aiding compliance and security monitoring.
* **Enhanced Data Discovery:** Easily search and explore available datasets across workspaces, accelerating collaboration and knowledge sharing.
* **Unified Governance:** Integrate with existing data catalogs and governance tools, maximizing your investments and simplifying data management.

**Use Cases:**

* **Simplifying Data Access Control:** Streamline user and group management, ensuring consistent access policies across your data lake.
* **Improving Data Quality:** Utilize data lineage to identify issues and dependencies, accelerating root cause analysis and ensuring data reliability.
* **Enhancing Security and Compliance:** Gain centralized visibility into data access and usage, facilitating regulatory compliance and security best practices.
* **Boosting Collaboration:** Empower data sharing and accelerate insights by allowing teams across workspaces to easily discover and access relevant datasets.

**Architectural Approach:**

* **Centralized Metastore:** Instead of individual metastores per workspace, Unity Catalog uses a single, shared repository for data definitions and metadata. This ensures consistency and eliminates siloed information.
* **Distributed Infrastructure:** The metastore itself is geographically distributed across multiple regions, offering high availability and scalability.
* **Microservices Architecture:** Unity Catalog utilizes a microservices architecture, decomposing functionalities into modular services for increased flexibility and resilience.

**Governance Principles:**

* **Role-Based Access Control (RBAC):** Granular control over data access is granted based on defined roles and permissions, ensuring data security and regulatory compliance.
* **Lineage Tracking:** Captures the transformation history of data across various stages, revealing dependencies and facilitating impact analysis.
* **Auditing and Logging:** Detailed records of data access and usage are captured for analysis and compliance purposes.
* **Data Discovery and Search:** Enables users to easily search and browse available datasets across workspaces, fostering collaboration and knowledge sharing.

**Data Lineage Benefits:**

* **Data Quality Management:** Identify data quality issues and root causes by tracing dependencies and transformations.
* **Impact Analysis:** Understand the downstream impact of changes made to data pipelines.
* **Auditability and Compliance:** Demonstrate data provenance and track regulatory compliance adherence.

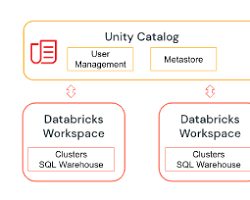
**Security Features:**

* **Encryption:** Data at rest and in transit is encrypted using industry-standard algorithms.
* **Authentication and Authorization:** Secure access to the metastore is enforced through robust authentication and authorization mechanisms.
* **Activity Monitoring:** Real-time monitoring of user activity and data access attempts ensures security and anomaly detection.

**Integration with Existing Solutions:**

* **Open APIs:** Unity Catalog offers open APIs for integrating with existing data catalogs, governance tools, and cloud platforms.
* **Connectors:** Pre-built connectors simplify integration with popular data storage systems and frameworks.

**Architecture diagram of Unity Catalog in Databricks:**

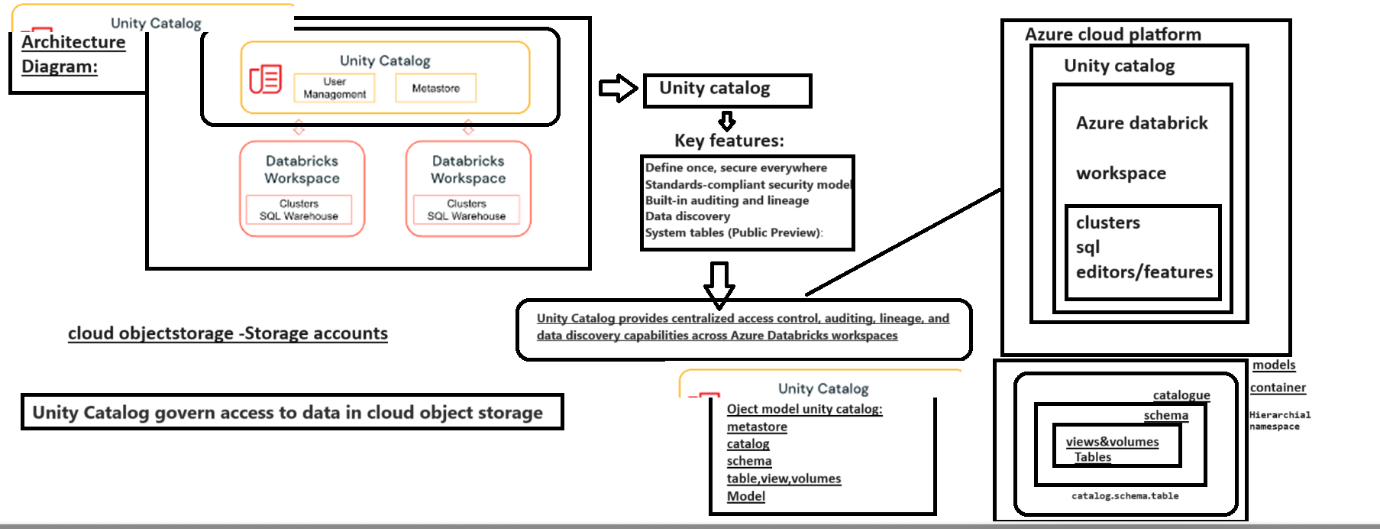


The diagram depicts the key components of Unity Catalog and how they interact with each other. As you can see, the architecture is built around a central metastore that stores all the metadata about your data assets. This metastore is accessed by various clients, such as notebooks, Spark jobs, and SQL queries, through a unified API. The API enforces access control and ensures that only authorized users can access data.

The lineage service tracks the lineage of your data, which means it keeps track of how your data has been transformed from its raw source to its final destination. This information is important for understanding the impact of changes to your data pipelines and for ensuring data quality.

The auditing service logs all access to the metastore and lineage service. This information can be used for security purposes and for compliance with regulations.

The Unity Catalog architecture is designed to be scalable and secure. It can be deployed on-premises or in the cloud, and it can handle large volumes of data.



**Unity Catalog object model in Databricks:**

Unity Catalog uses a hierarchical object model with three primary levels:

1. **Metastore:** The top-level container for metadata, providing a single source of truth for data definitions and access control across your workspaces.
2. **Catalog:** Represents a logical grouping of data assets within a metastore. You can organize catalogs based on project, department, or other criteria.
3. **Schema (Database):** Contains tables, views, and volumes within a catalog. Think of it as a logical container for related data assets.
4. **Table/View/Volume:** The atomic units of data storage.
   * Tables store structured data.
   * Views offer virtual representations of underlying data without physical storage.
   * Volumes manage non-tabular data like images or documents.

This hierarchical structure offers clear organization and granular access control across your data lake in Databricks. With this understanding, you can effectively navigate and manage your data using the Unity Catalog features.

**Central Metadata Store:**

* At the core lies the **central metadata store**, a distributed system responsible for storing all metadata about your data assets. This store ensures consistency and eliminates siloed information across workspaces.

**Object Hierarchy:**

* Three primary levels exist:
  + **Metastore:** The top level, housing multiple catalogs.
  + **Catalog:** Groups data assets based on projects, departments, etc.
  + **Schema (Database):** Contains tables, views, and volumes within a catalog.
  + **Table/View/Volume:** Represent individual data storage units.

**Microservices Architecture:**

* Unity Catalog utilizes a **microservices architecture**, breaking down functionalities into modular services for flexibility and resilience. These services include:
  + **Catalog Service:** Manages catalogs, schemas, and tables.
  + **Lineage Service:** Tracks data transformation history.
  + **Access Control Service:** Enforces permission checks.
  + **Auditing Service:** Logs access and usage data.

**Centralized API:**

* A single, **unified API** provides access to all services, offering clients (notebooks, Spark jobs, SQL queries) a consistent way to interact with the object model.

**Clients and Interactions:**

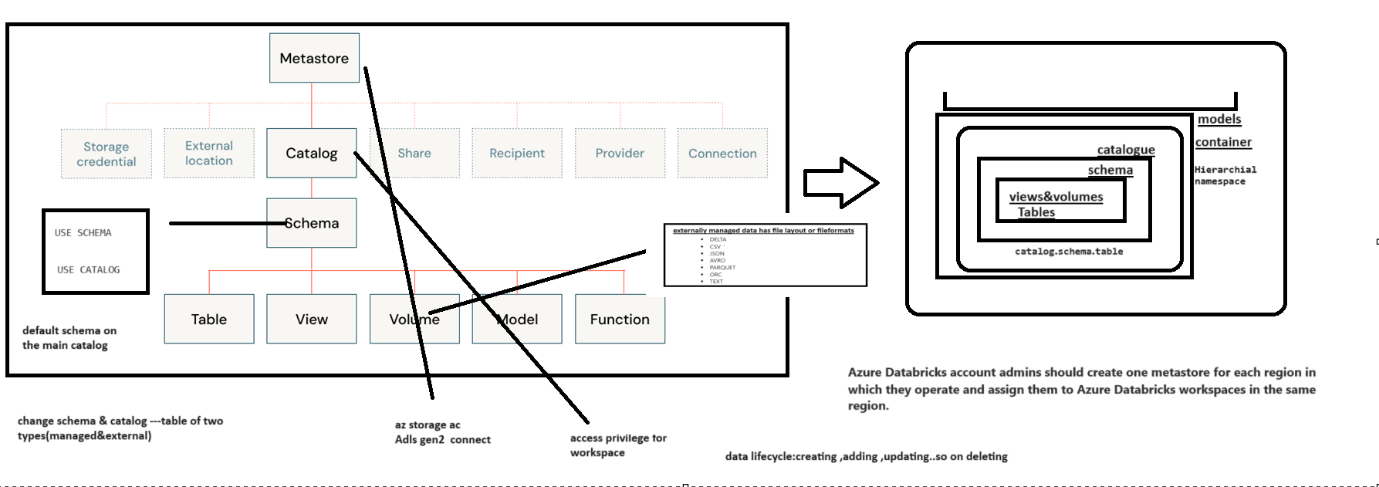
* Clients issue requests through the API.
* The API directs requests to relevant services.
* Services process requests and return results or updates.

**Security and Scalability:**

* Secure communication channels protect data access.
* Distributed infrastructure ensures high availability and scalability for handling large data volumes.

**Key Features within the Architecture:**

* **Open APIs:** Support integration with existing data catalogs, governance tools, and cloud platforms.
* **Connectors:** Pre-built connectors simplify integration with popular data storage systems and frameworks.
* **Lineage Support:** Tracks data transformations for impact analysis and quality management.
* **Fine-grained Access Control:** Grants permissions at various levels (catalogs, schemas, tables) for enhanced data security.



# **unity-catalog-quickstart-python(Python)**

# Create a catalog

To create a catalog, use the CREATE CATALOG command with spark.sql. You must be a metastore admin or user with the CREATE CATALOG privilege on the metastore to create a catalog. If your workspace was enabled for Unity Catalog by default, then workspace admins have the CREATE CATALOG privilege by default.

If your workspace was enabled for Unity Catalog by default, then there may be no managed storage location for the metastore, and you must create a location for the new catalog.

The following commands show how to:

1. Create a catalog.
2. Select a catalog.
3. Show all catalogs.
4. Grant permissions on a catalog.
5. Show all grants on a catalog.

# Create a catalog.

spark.sql("CREATE CATALOG IF NOT EXISTS quickstart\_catalog")

# Create a catalog and specify the managed location

# spark.sql("CREATE CATALOG IF NOT EXISTS quickstart\_catalog MANAGED LOCATION '<location-path>'")

# Set the current catalog.

spark.sql("USE CATALOG quickstart\_catalog")

# Show all catalogs in the metastore.

display(spark.sql("SHOW CATALOGS"))

# Grant create and use catalog permissions for the catalog to all users on the account.

# This also works for other account-level groups and individual users.

spark.sql("""

GRANT CREATE, USE CATALOG

ON CATALOG quickstart\_catalog

TO `account users`""")

# Show grants on the quickstart catalog.

display(spark.sql("SHOW GRANT ON CATALOG quickstart\_catalog"))

# **Create and manage schemas (databases)**

Schemas, also called databases, are the second level of the Unity Catalog three-level namespace. Schemas logically organize tables and views.

# Create a schema in the catalog that was set earlier.

spark.sql("""

CREATE SCHEMA IF NOT EXISTS quickstart\_schema

COMMENT 'A new Unity Catalog schema called quickstart\_schema'""")

# Show schemas in the catalog that was set earlier.

display(spark.sql("SHOW SCHEMAS"))

# Describe the schema.

display(spark.sql("DESCRIBE SCHEMA EXTENDED quickstart\_schema"))

# Grant create table, and use schema permissions for the schema to all users on the account.

# This also works for other account-level groups and individual users.

spark.sql("""

GRANT CREATE TABLE, USE SCHEMA

ON SCHEMA quickstart\_schema

TO `account users`""")

# **Create a managed table**

Managed tables are the default way to create table with Unity Catalog. The table is created in the managed storage location configured for the metastore, catalog, or schema.

The following commands show how to:

1. Select a schema.
2. Create a managed table and insert records into it.
3. Show all tables in a schema.
4. Describe a table.

# Set the current schema.

spark.sql("USE quickstart\_schema")

# Show the current database (also called a schema).

spark.catalog.currentDatabase()

# Create a managed Delta table in the catalog that was set earlier.

spark.sql("CREATE OR REPLACE TABLE quickstart\_table (id STRING)")

# Grant select and modify permissions for the table to all users on the account.

# This also works for other account-level groups and individual users.

spark.sql("""

GRANT SELECT, MODIFY

ON TABLE quickstart\_table

TO `account users`""")

# List the available tables in the catalog that was set earlier.

display(spark.sql("SHOW TABLES"))

# Insert 10 rows into the table.

spark.range(10).selectExpr("id").write.insertInto("quickstart\_table")

# Show the table.

display(spark.table("quickstart\_table"))

# Show all of the available tables in the schema.

display(spark.sql("SHOW TABLES in quickstart\_schema"))

1. **Execute & explain, Azure datafactory and its copy activity.**

**What is Azure Data Factory?**

Azure Data Factory is a cloud-based data integration service that allows you to create data-driven workflows in the cloud for orchestrating and automating data movement and data transformation.

ADF does not store any data itself. It allows you to create data-driven workflows to orchestrate the movement of data between supported data stores and then process the data using compute services in other regions or in an on-premise environment. It also allows you to monitor and manage workflows using both programmatic and UI mechanisms.

**Azure Data Factory use cases**

ADF can be used for:

* Supporting data migrations
* Getting data from a client’s server or online data to an Azure Data Lake
* Carrying out various data integration processes
* Integrating data from different ERP systems and loading it into Azure Synapse for reporting

**How does Azure Data Factory work?**

The Data Factory service allows you to create data pipelines that move and transform data and then run the pipelines on a specified schedule (hourly, daily, weekly, etc.). This means the data that is consumed and produced by workflows is time-sliced data, and we can specify the pipeline mode as scheduled (once a day) or one time.

Azure Data Factory pipelines (data-driven workflows) typically perform three steps.

**Step 1: Connect and Collect**

Connect to all the required sources of data and processing such as SaaS services, file shares, FTP, and web services. Then,  move the data as needed to a centralized location for subsequent processing by using the Copy Activity in a data pipeline to move data from both on-premise and cloud source data stores to a centralization data store in the cloud for further analysis.

**Step 2: Transform and Enrich**

Once data is present in a centralized data store in the cloud, it is transformed using compute services such as HDInsight Hadoop, Spark, Azure Data Lake Analytics, and Machine Learning.

**Step 3: Publish**

Deliver transformed data from the cloud to on-premise sources like SQL Server or keep it in your cloud storage sources for consumption by BI and analytics tools and other applications.

**Data migration activities with Azure Data Factory**

By using Microsoft Azure Data Factory, data migration occurs between two cloud data stores and between an on-premise data store and a cloud data store.

*Copy Activity* in Azure Data Factory copies data from a source data store to a sink data store. Azure supports various data stores such as source or sink data stores like Azure Blob storage, Azure Cosmos DB (DocumentDB API), Azure Data Lake Store, Oracle, Cassandra, etc. For more information about Azure Data Factory supported data stores for data movement activities, refer to Azure documentation for data movement activities.

Azure Data Factory supports transformation activities such as Hive, MapReduce, Spark, etc that can be added to pipelines either individually or chained with other activities. For more information about ADF-supported data stores for data transformation activities, refer to the following Azure Data Factory documentation: Transform data in Azure Data Factory.

If you want to move data to/from a data store that Copy Activity doesn’t support, you should use a .NET custom activity in Azure Data Factory with your own logic for copying/moving data. To learn more about creating and using a custom activity, check the Azure documentation and see “Use custom activities in an Azure Data Factory pipeline”.

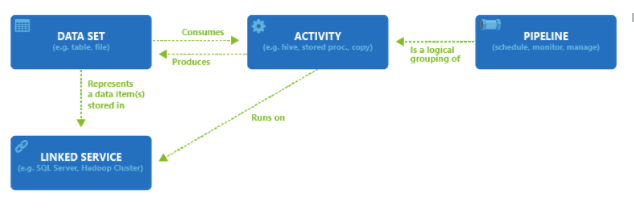
**Azure Data Factory key components**

Azure Data Factory has four key components that work together to define input and output data, processing events, and the schedule and resources required to execute the desired data flow:

* **Datasets represent data structures within the data stores.** An input dataset represents the input for an activity in the pipeline. An output dataset represents the output for the activity. For example, an Azure Blob dataset specifies the blob container and folder in the Azure Blob Storage from which the pipeline should read the data. Or, an Azure SQL Table dataset specifies the table to which the output data is written by the activity.
* **A pipeline is a group of activities.** They are used to group activities into a unit that together performs a task. A data factory may have one or more pipelines. For example, a pipeline could contain a group of activities that ingests data from an Azure blob and then runs a Hive query on an HDInsight cluster to partition the data.
* **Activities define the actions to perform on your data.** Currently, Azure Data Factory supports two types of activities: data movement and data transformation.
* **Linked services define the information needed for Azure Data Factory to connect to external resources.** For example, an Azure Storage linked service specifies a connection string to connect to the Azure Storage account.

**How the Azure Data Factory components work together**

The following schema shows us the relationships between the **Dataset**, **Activity**, **Pipeline**, and **Linked Services** components:



**Azure Data Factory access zones**

Currently, you can create data factories in the West US, East US, and North Europe regions. However, a data factory can access data stores and compute services in other Azure regions to move data between data stores or process data using compute services.  
For example, let’s say that your compute environments such as Azure HDInsight cluster and Azure Machine Learning are running out of the West Europe region. You can create and use an Azure Data Factory instance in North Europe and use it to schedule jobs on your compute environments in West Europe. It takes a few milliseconds for Data Factory to trigger the job on your compute environment but the time for running the job on your computing environment does not change.

You can use one of the following tools or APIs to create data pipelines in Azure Data Factory:

* Azure portal
* Visual Studio
* PowerShell
* .NET API
* REST API
* Azure Resource Manager template

**Data Migration in action**

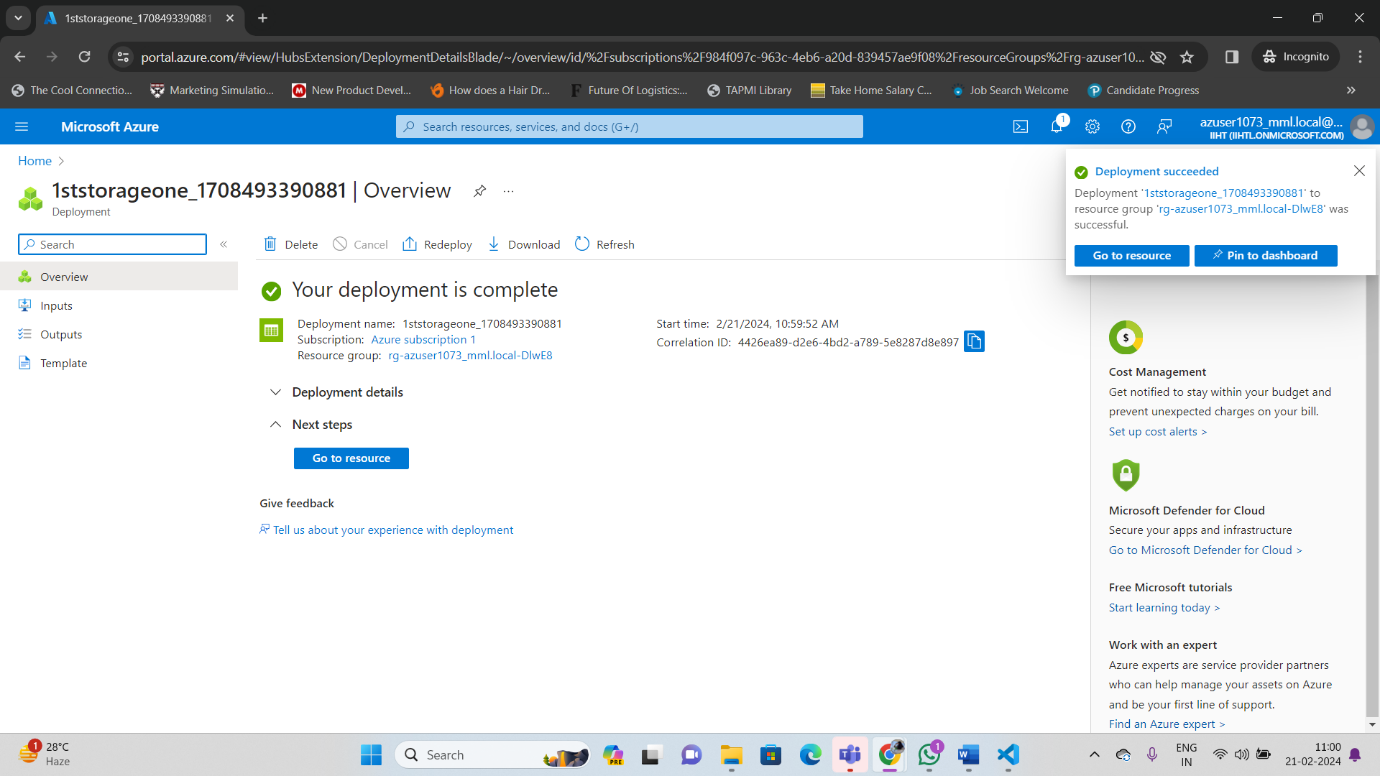
To get started with Data Factory, you should create a Data Factory on Azure, then create the four key components with Azure Portal, Virtual Studio, or PowerShell etc. Since the four components are in editable JSON format, you can also deploy them in a whole ARM template on the fly.

**DataCopy Wizard on Azure**

To start migrating the data on Blob storage to Azure SQL, the most simple way is to use Data Copy Wizard, which is currently in preview. It allows you to quickly create a data pipeline that copies data from a supported source data store to a supported destination data store. For more information on creating your migration related components with Data Copy Wizard, Create a pipeline with Copy Activity using Data Factory Copy Wizard.

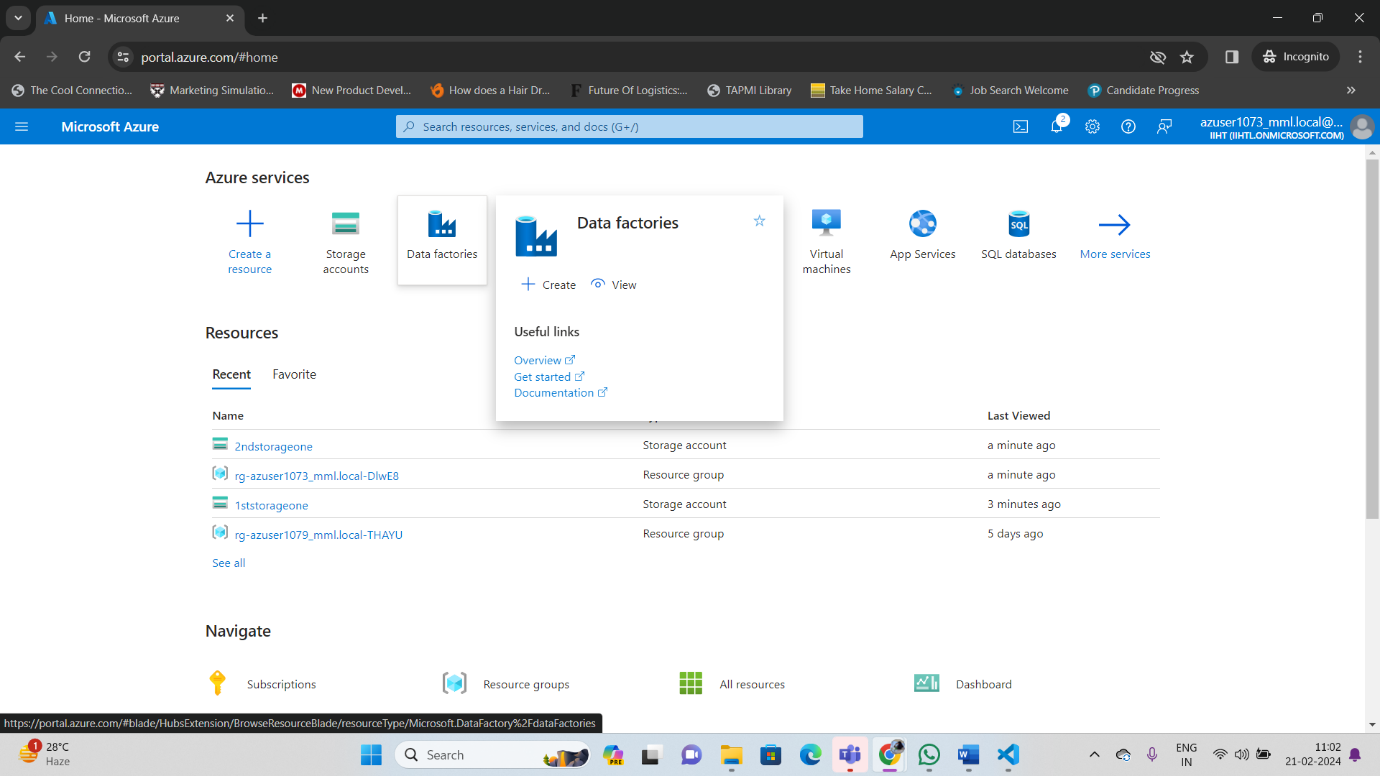
**Hands On Practice:**

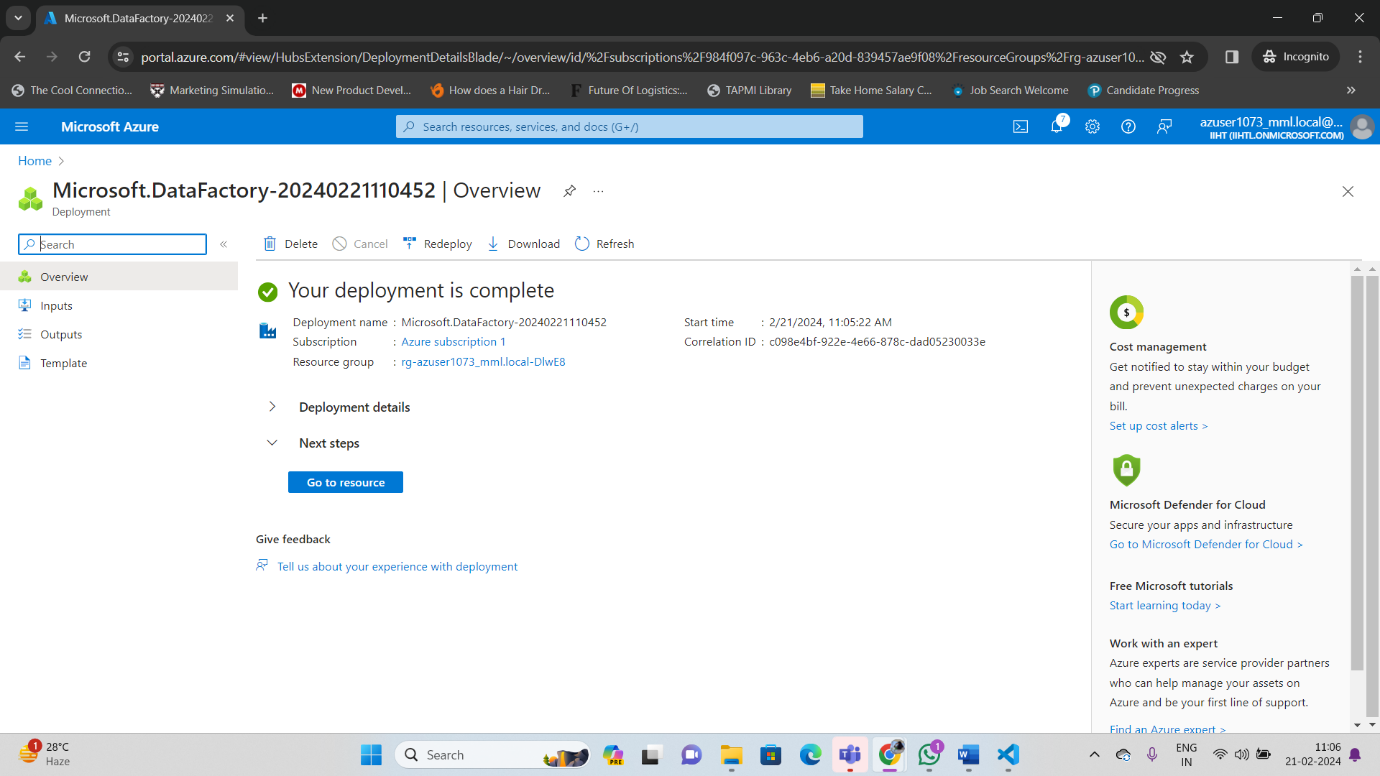
**Creating Storage accounts :**

****

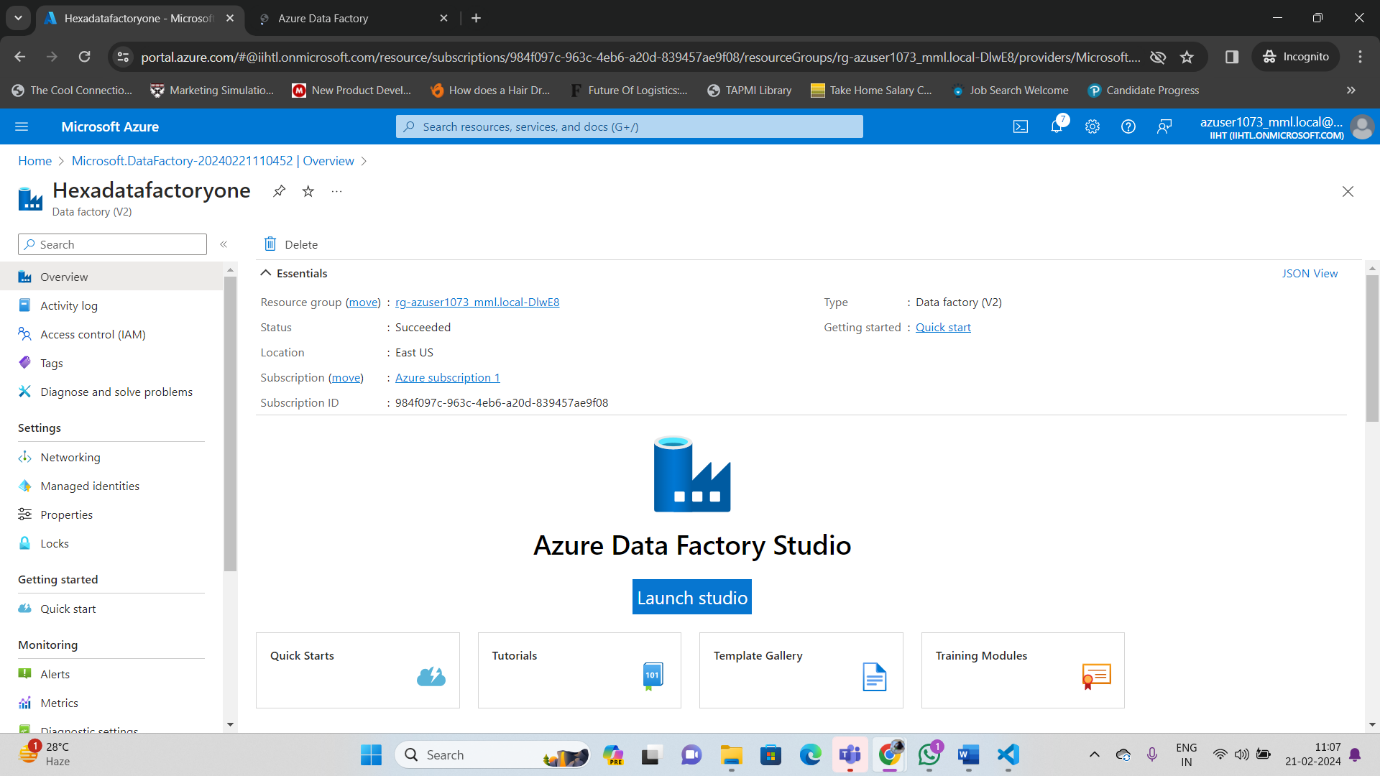
****

**Creating account in datafactory :**

****

****

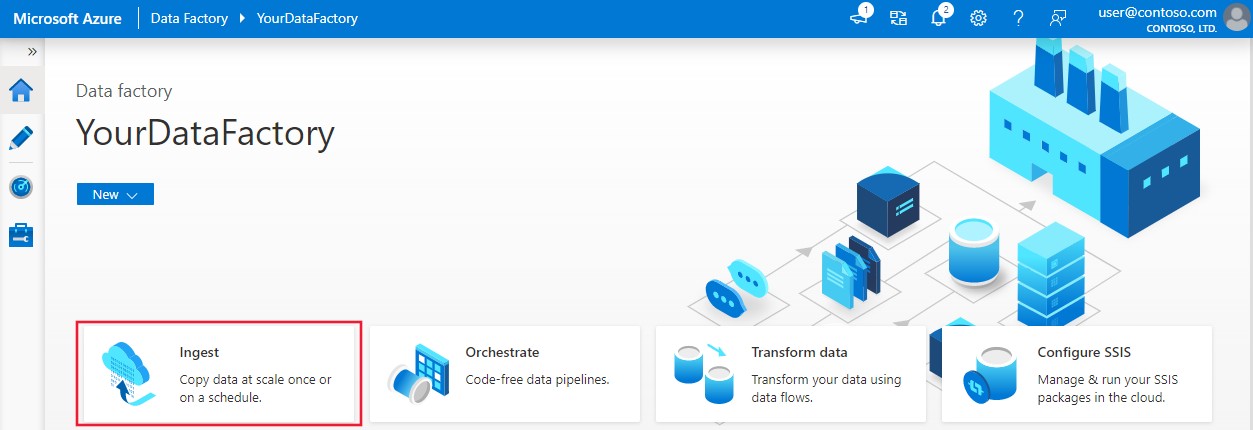
**Launch studio :**

****

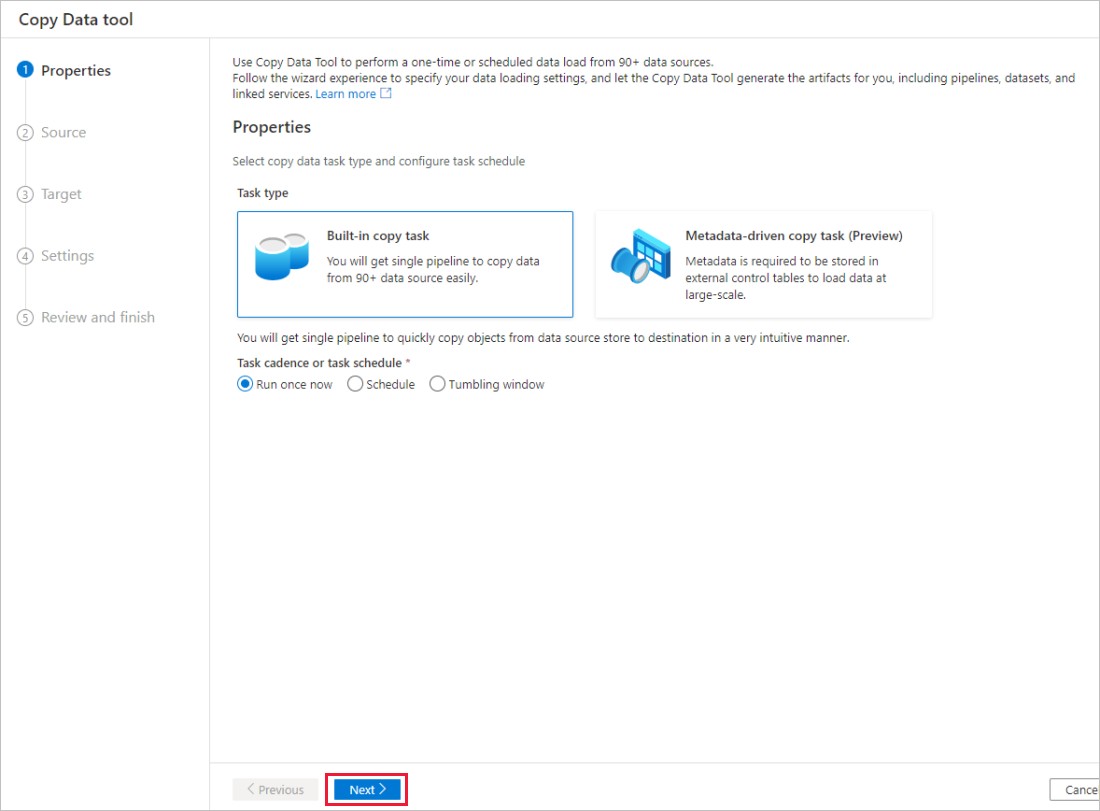
**Create a data factory**

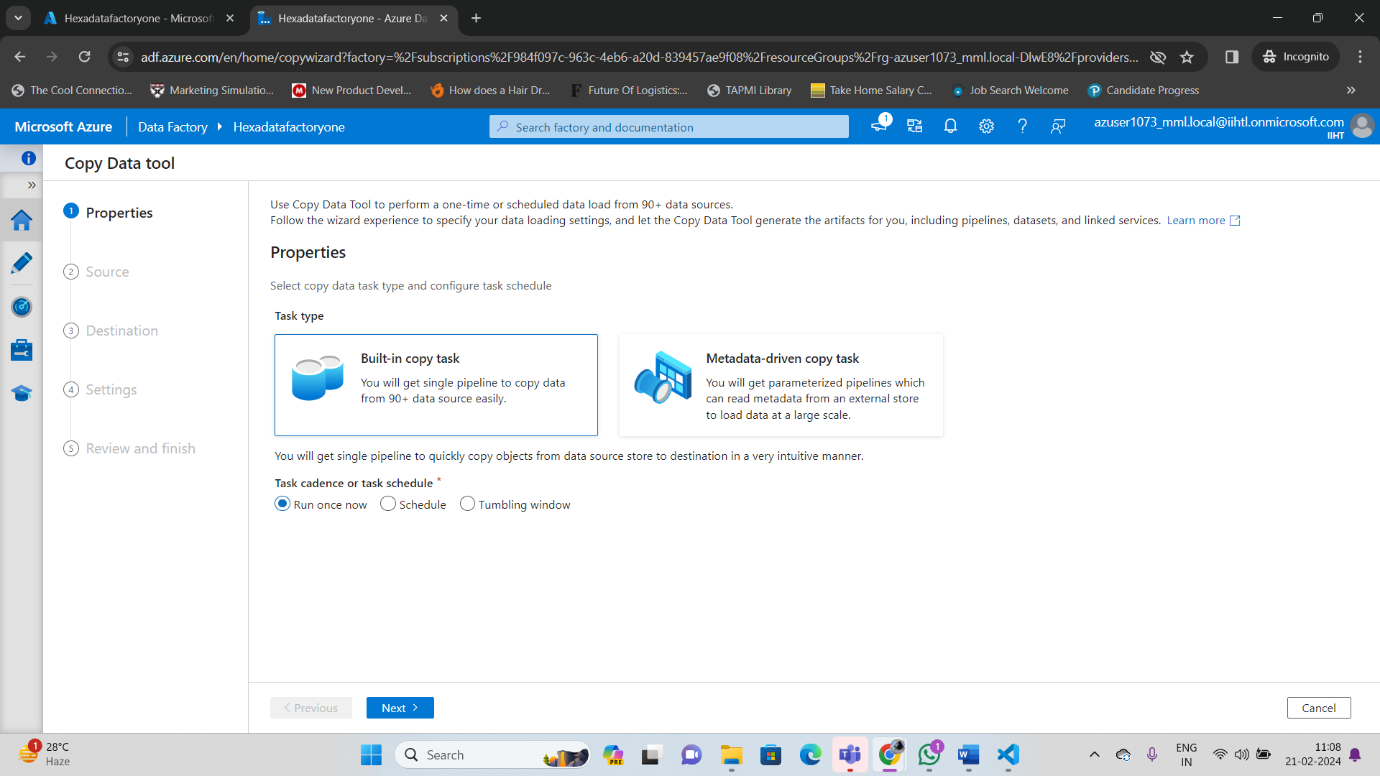
You can use your existing data factory or create a new one as described in Quickstart: Create a data factory by using the Azure portal. Use the copy data tool to copy data The steps below will walk you through how to easily copy data with the copy data tool in Azure Data Factory.

Step 1: Start the copy data Tool 1. On the home page of Azure Data Factory, select the Ingest tile to start the Copy Data tool.



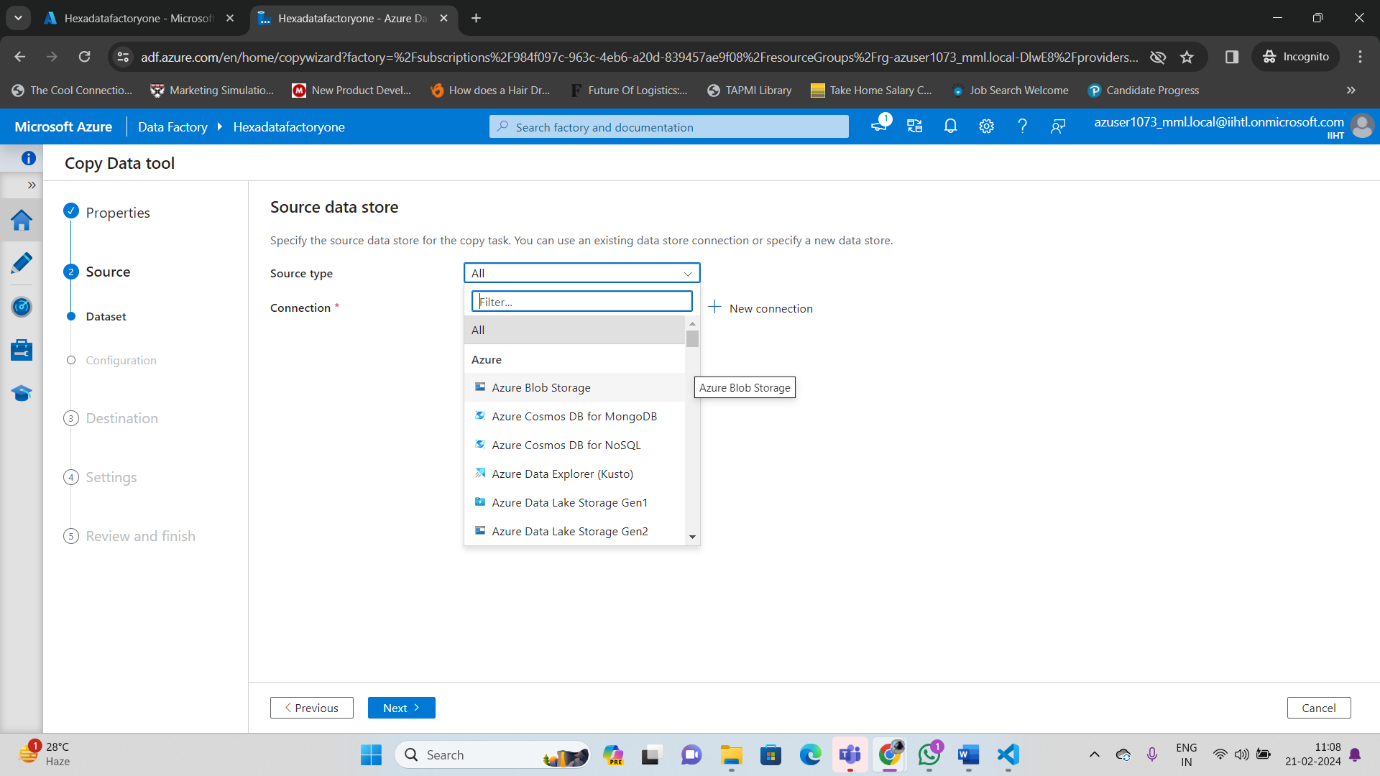
On the **Properties** page of the Copy Data tool, choose **Built-in copy task** under **Task type**, then select **Next**.

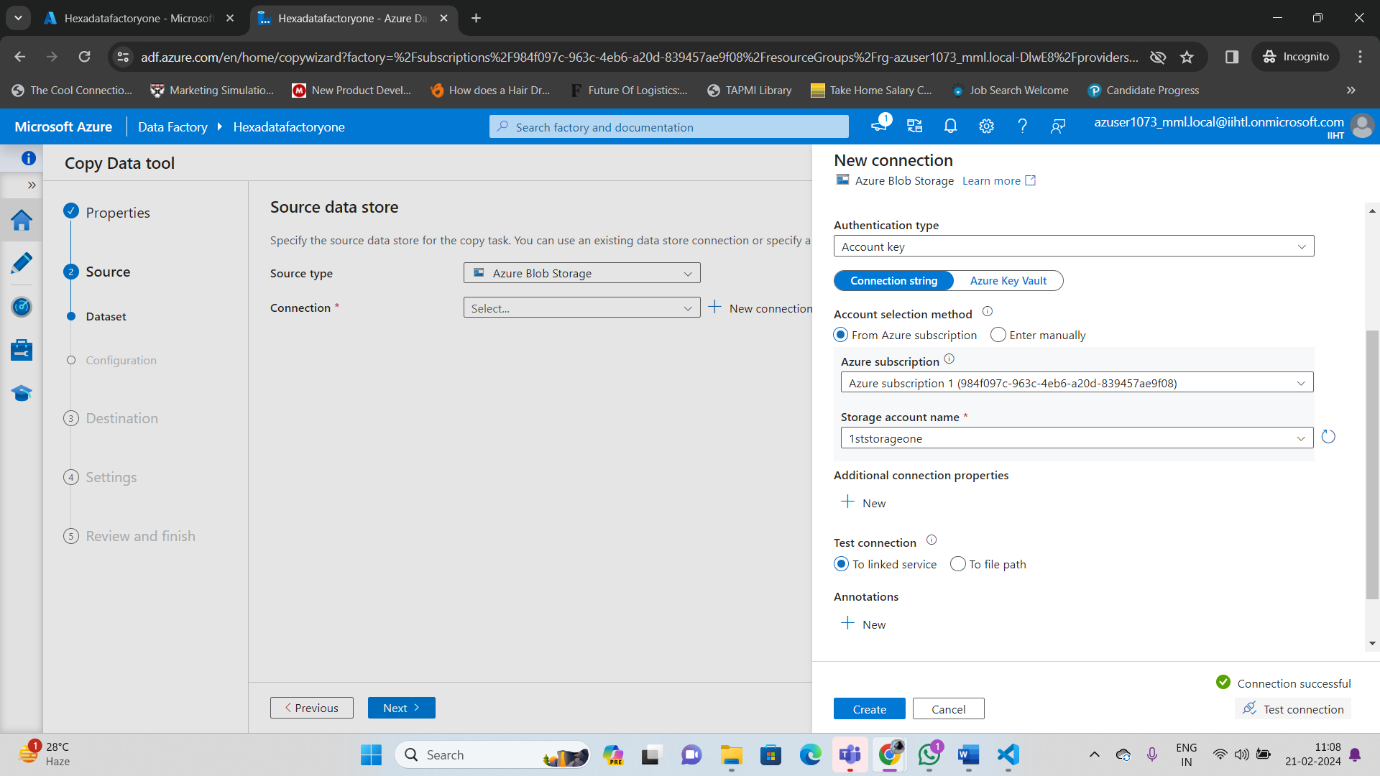


****

# Step 2: Complete source conﬁguration

1. Click **+ Create new connection** to add a connection.
2. Select the linked service type that you want to create for the source connection. In this tutorial, we use **Azure Blob Storage**. Select it from the gallery, and then select **Continue**.





Select the newly created connection in the **Connection** block.

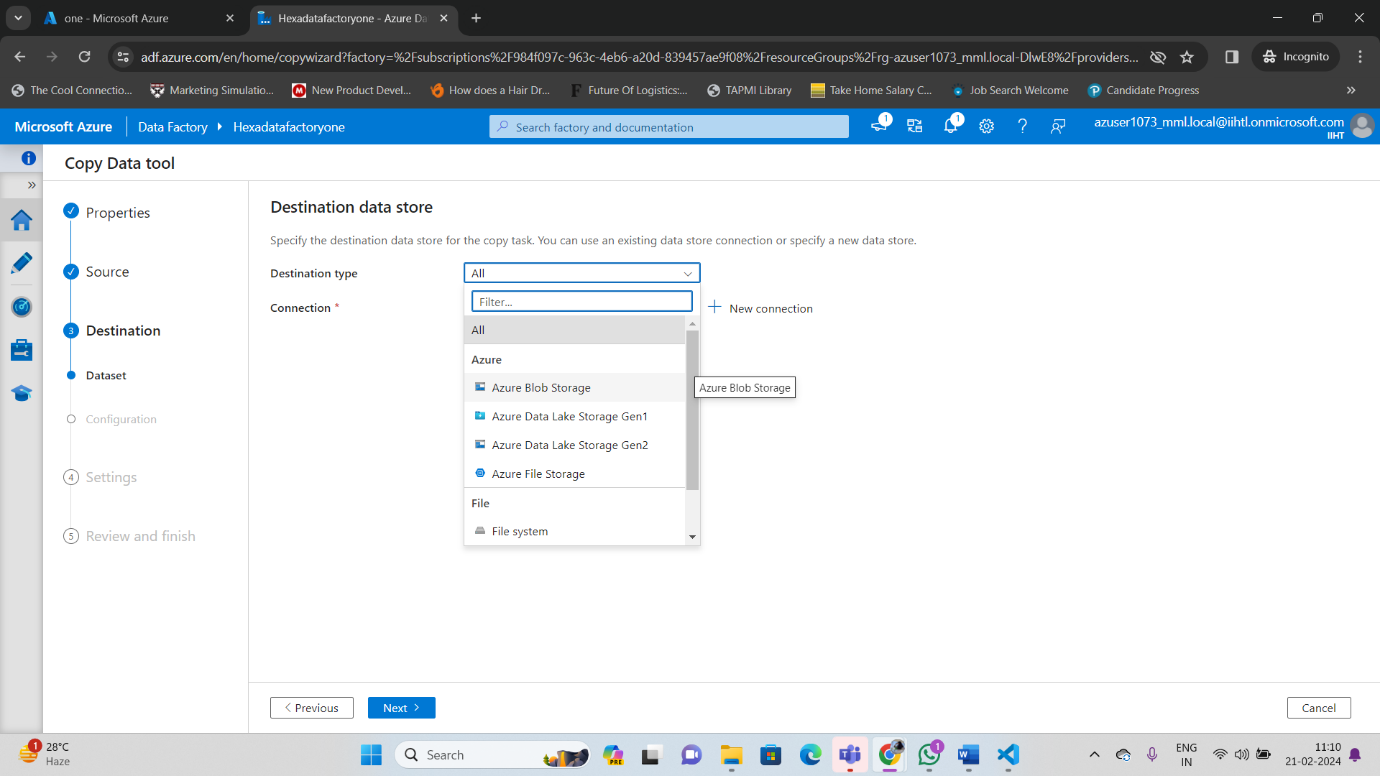
In the **File or folder** section, select **Browse** to navigate to the **adftutorial/input** folder, select the **emp.txt** ﬁle, and then click **OK**.

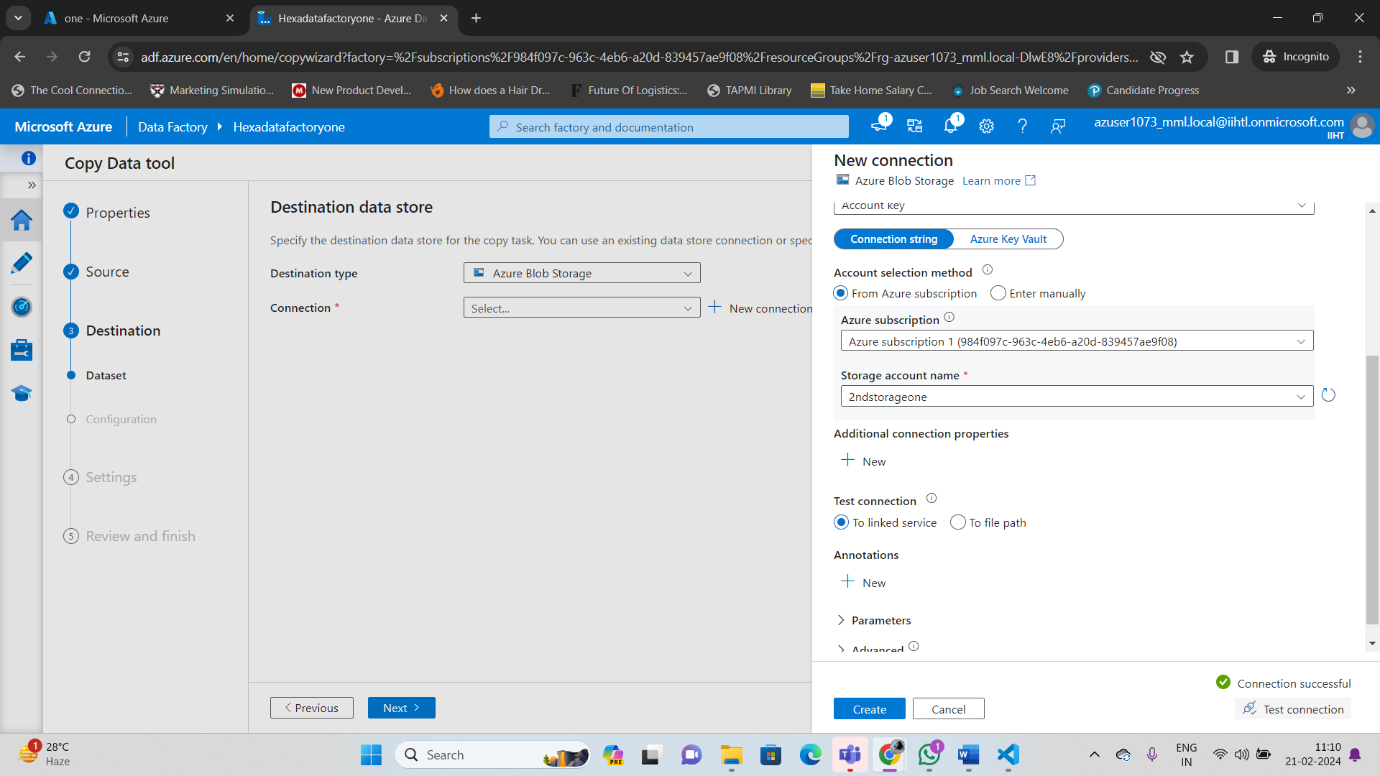
Select the **Binary copy** checkbox to copy ﬁle as-is, and then select **Next**.

****

# Step 3: Complete destination conﬁguration

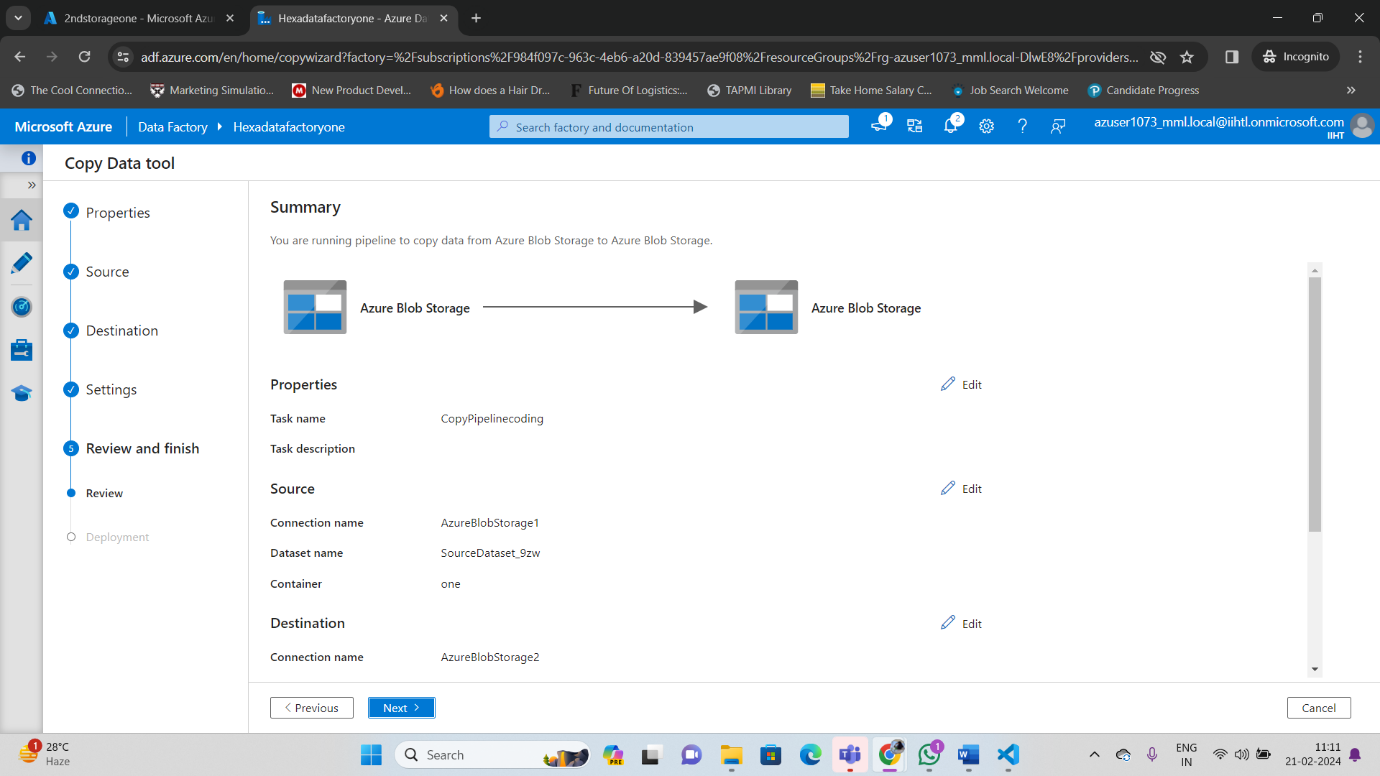
1. Select the **AzureBlobStorage** connection that you created in the **Connection** block.
2. In the **Folder path** section, enter **adftutorial/output** for the folder path.

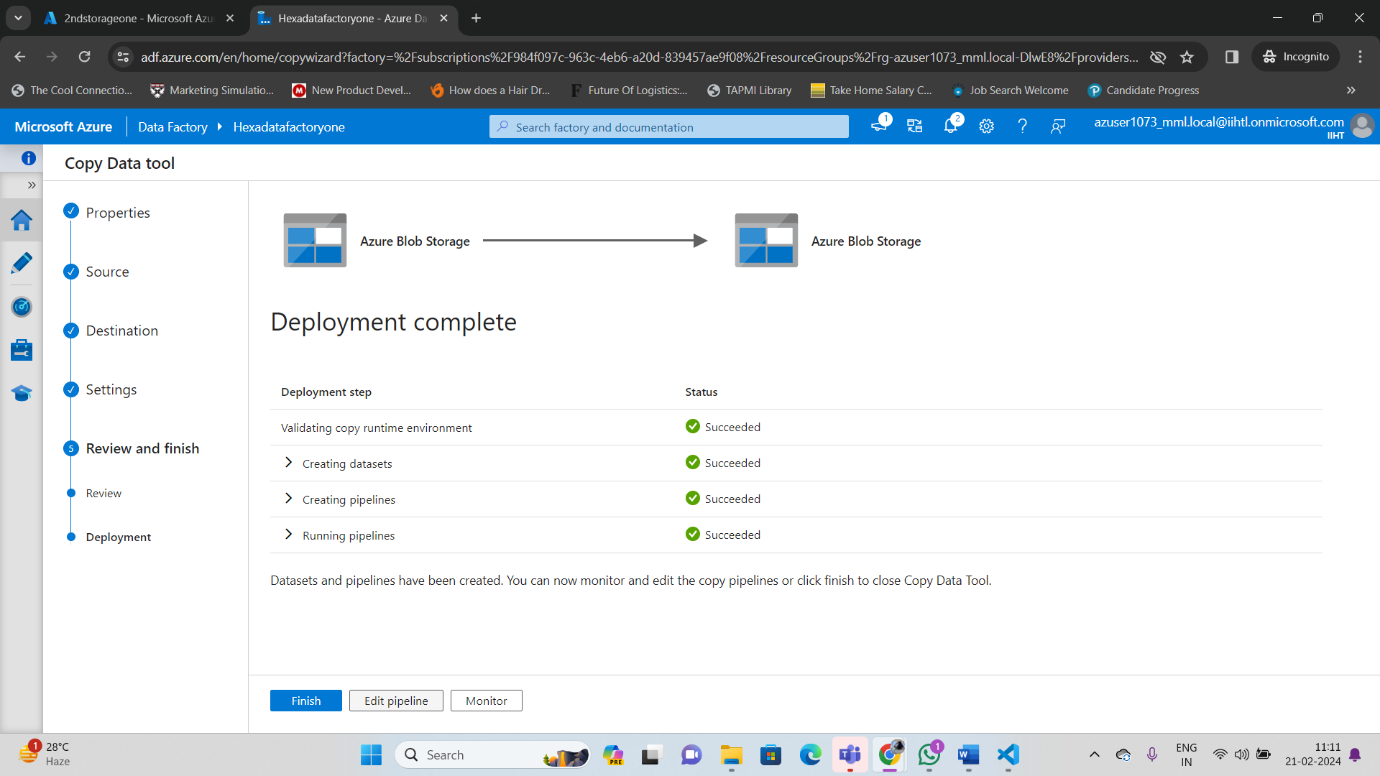
****

****

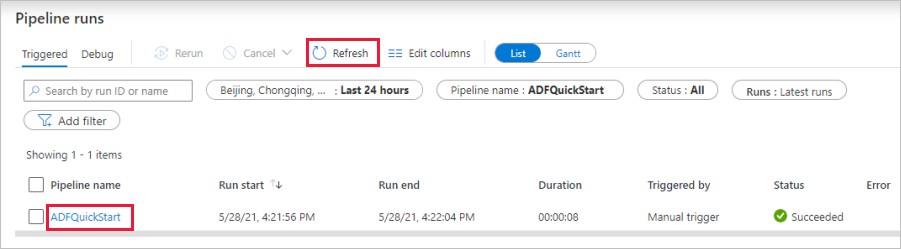
# Step 4: Review all settings and deployment

1. On the **Settings** page, specify a name for the pipeline and its description, then select **Next** to use other default conﬁgurations.
2. On the **Summary** page, review all settings, and select **Next**.
3. On the **Deployment complete** page, select **Monitor** to monitor the pipeline that you created.

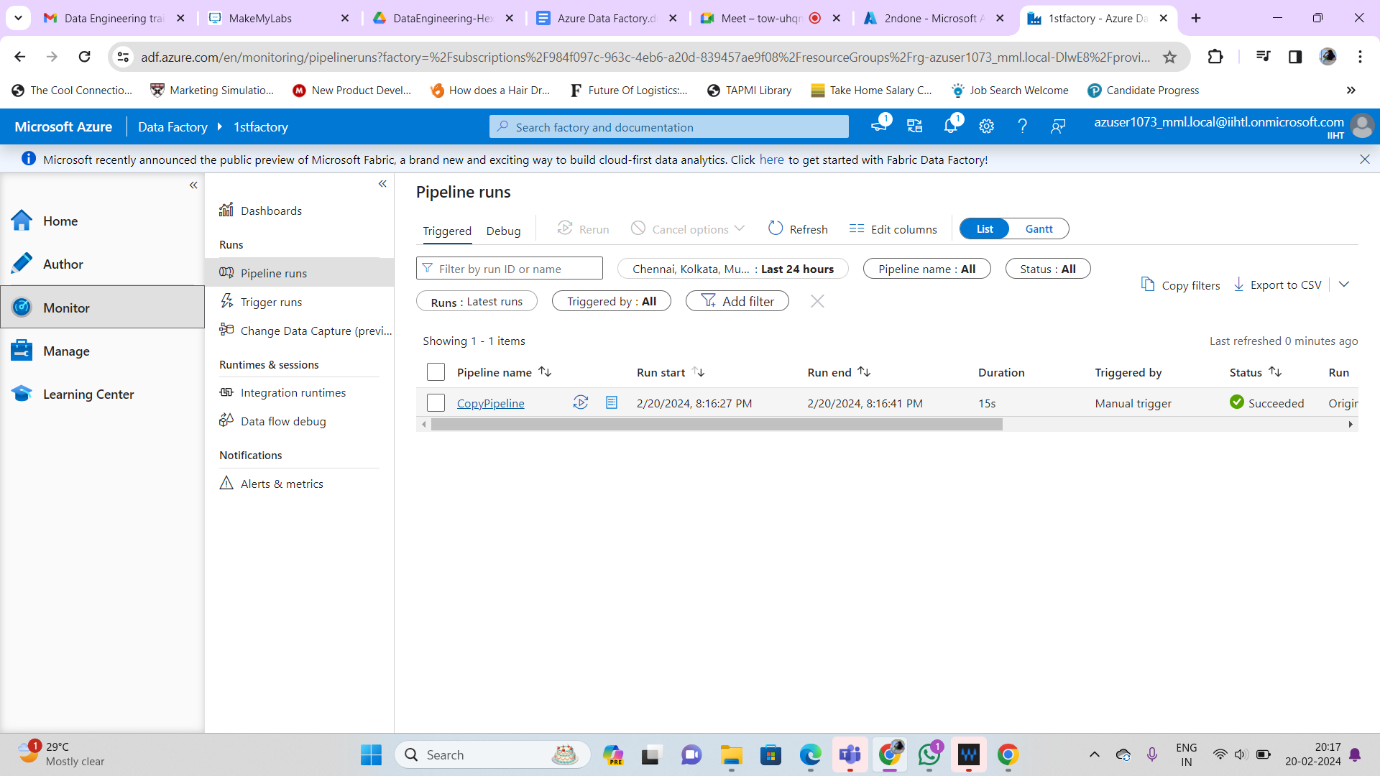


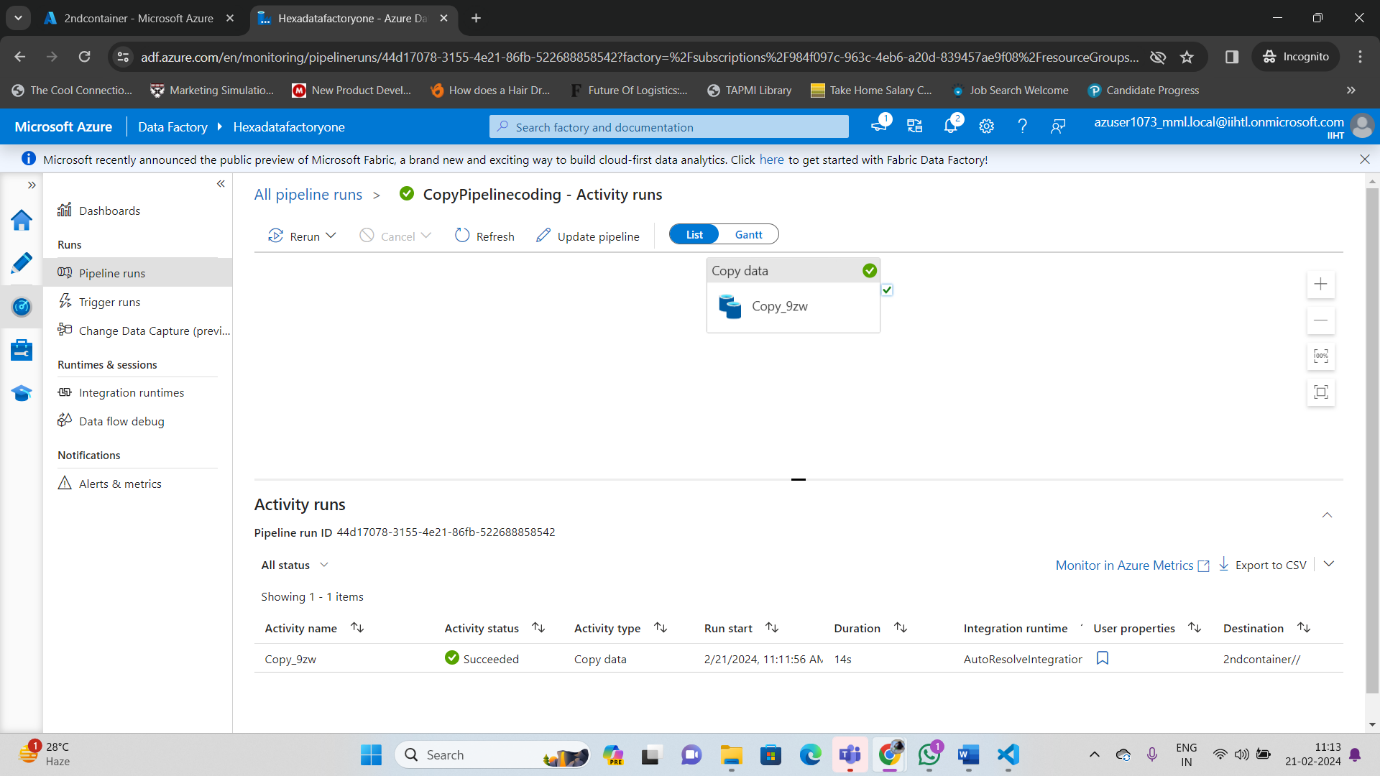
****

# Step 5: Monitor the running results

1. The application switches to the **Monitor** tab. You see the status of the pipeline on this tab. Select **Refresh** to refresh the list. Click the link under **Pipeline name** to view activity run details or rerun the pipeline.
2. On the Activity runs page, select the **Details** link (eyeglasses icon) under the **Activity name** column for more details about copy operation. For details about the properties, see Copy Activity overview.

**Check Pipeline status**

****

****

**Check data in both container**

The files in the 1st storage container has been copied to 2nd storage container.



