Microsoft

Microsoft Partner Project Ready

Implement with Impact

Modern Data Platform with Azure Databricks

<Speaker name or subtitle>

<Date>

Day 2 of 3

Technofocus

Course Plan and Learning Objectives

Day 1

Module 1 - Introduction to Azure Databricks
- Azure Databricks: A Data Intelligent Platform
- Why Azure Databricks
- Decision guide: Azure Databricks vs. Microsoft Fabric

Module 2 - Migration to Azure Databricks
- Microsoft Cloud Adoption Framework for Azure
- Migration strategies
- Data landing zones
- Migration scenarios

Interactive Simulated Lab Experience
- End-to-End Streaming Pipeline with Lakeflow Declarative Pipelines in Azure Databricks

Day 2

Module 3 - Integration with Azure
- Seamless integration with Microsoft Azure services
- Connect to Azure Data Lake Storage (ADLS) Gen2 and Blob Storage
- Leverage Azure Databricks for Azure Cosmos DB Operations
- Secret management with Azure Key Vault
- Connect Azure Databricks to Azure Event Hubs

Module 4 - Integration with Microsoft Fabric and Power BI
- Data Intelligence with Azure Databricks and Microsoft Fabric
- Connect Power BI to Azure Databricks
- Integration with Azure Data Factory
- Mirroring Azure Databricks Unity Catalog

Interactive Simulated Lab Experience
- Setup and use Unity Catalog for Data Management in Azure Databricks
- Real-Time Streaming with Azure Databricks and Azure Event Hubs

Day 3

Module 5 - Integration with Azure AI Foundry
- Azure Databricks connector in Azure AI Foundry
- Mosaic AI and machine learning on Azure Databricks
- Query Generative AI model serving endpoints
- Databricks Assistant, AI/BI Genie and AI Functions on Azure Databricks
- Chat with LLMs and prototype GenAI apps using AI Playground
- Build and optimize agents on your data with Agent Bricks

Module 6 - Security and Governance
- Integrate Azure Databricks with Microsoft Purview
- Integration of Azure Databricks Unity Catalog with Microsoft Purview

Module 7 - Well-architected for Azure Databricks
- Lakehouse implementation: Principles and best practices
- Azure Databricks well-architected framework

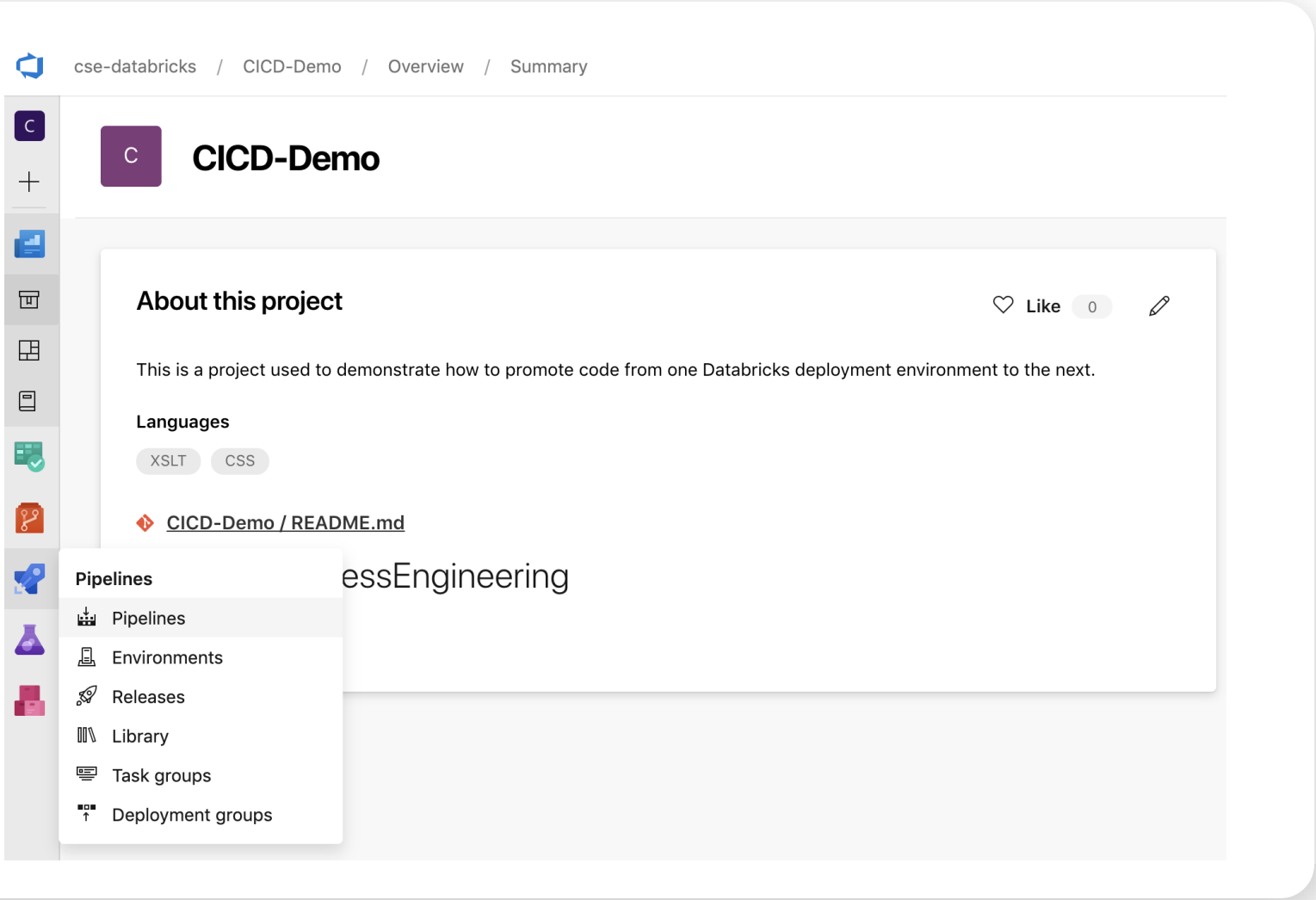
Interactive Simulated Lab Experience
- Responsible AI with Large Language Models using Azure Databricks and Azure OpenAI
- Connect to and manage Azure Databricks in Microsoft Purview

03
Integration with Azure

Seamless integration with Microsoft Azure services

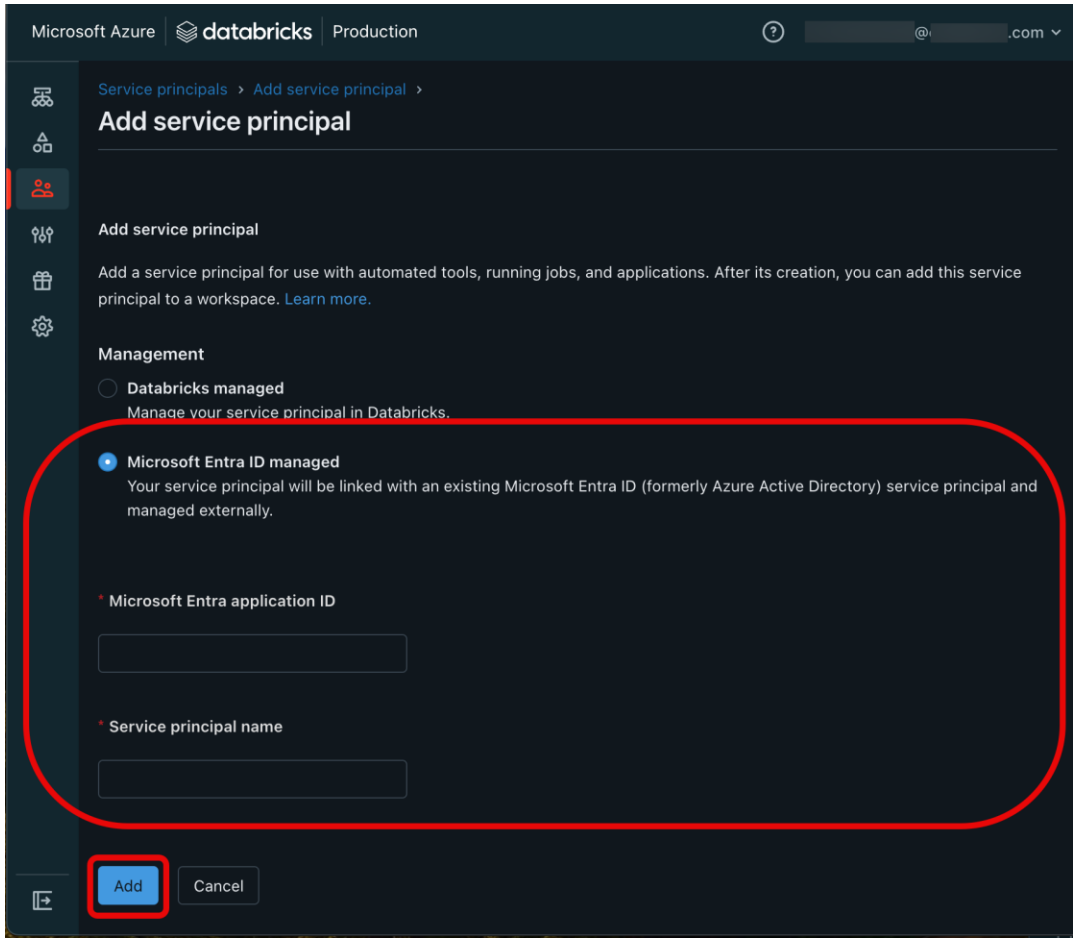
Integration with Azure DevOps

- Azure Databricks connects with Azure DevOps to help enable Continuous Integration and Continuous Deployment (CI/CD)



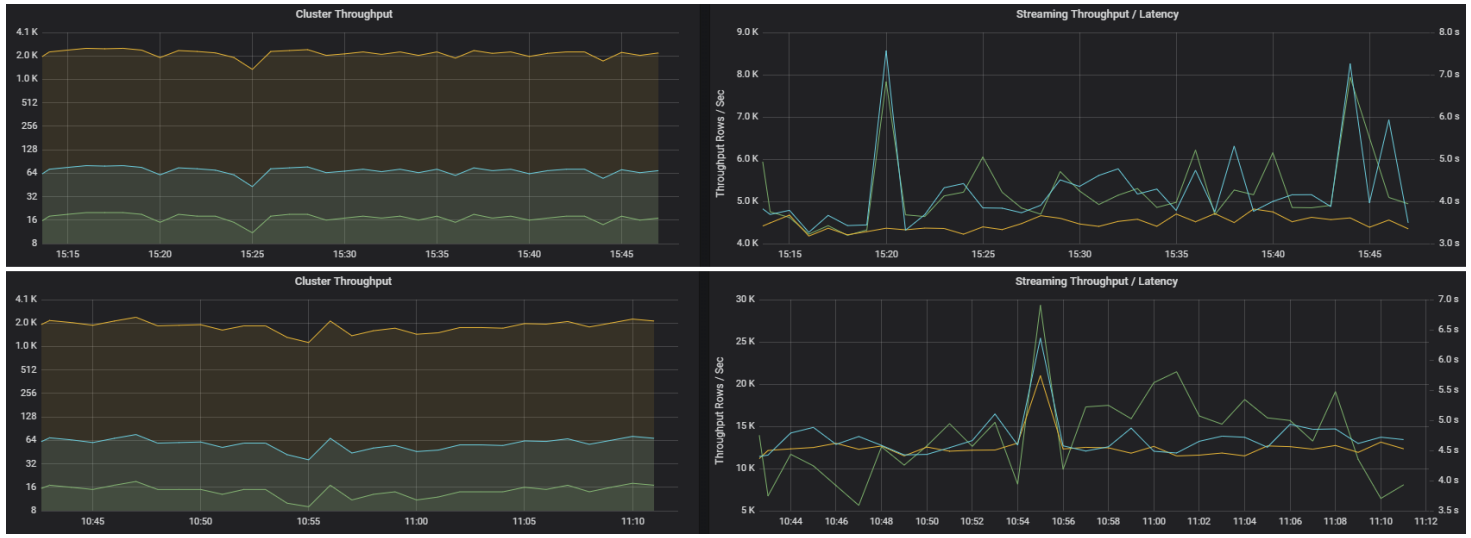
Enable conditional access for Azure Databricks with Microsoft Entra ID

- Azure Databricks supports Microsoft Entra ID conditional access
- This allows administrators to control where and when users are permitted to sign in to Azure Databricks
- Sync from Entra using SCIM
- Automatic Identity Management



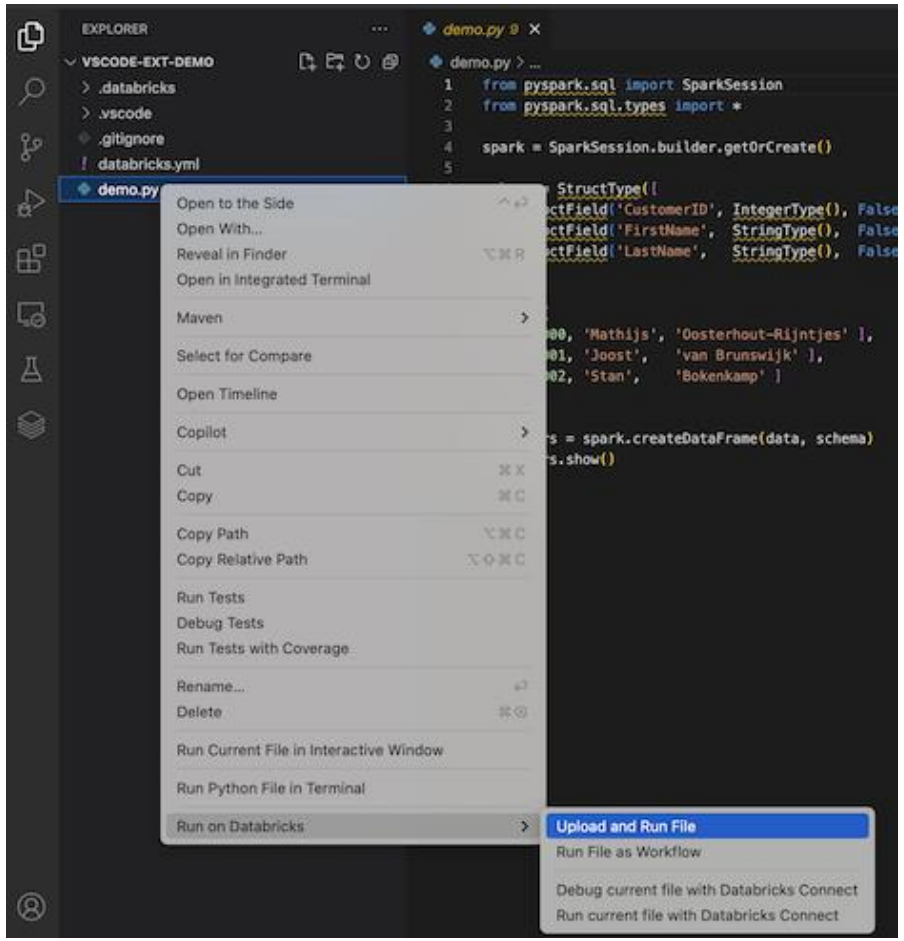
Monitor Azure Databricks jobs with Azure Monitor

- Azure Databricks offers robust functionality for monitoring
  - Custom application metrics
  - Streaming query events
  - Application log messages
- Azure Databricks can send this monitoring data to Azure Monitor



Azure Databricks extension for Visual Studio Code

- Enables you to connect to your remote Azure Databricks workspaces from the Visual Studio Code integrated development environment (IDE) running on your local development machine



Connect to Azure Data Lake Storage (ADLS) Gen2 and Blob Storage

Integrate Azure Databricks with ADLS Gen2

- Azure Databricks can directly read and write data from ADLS Gen2 for fastest possible data access, enabling efficient data processing and analytics

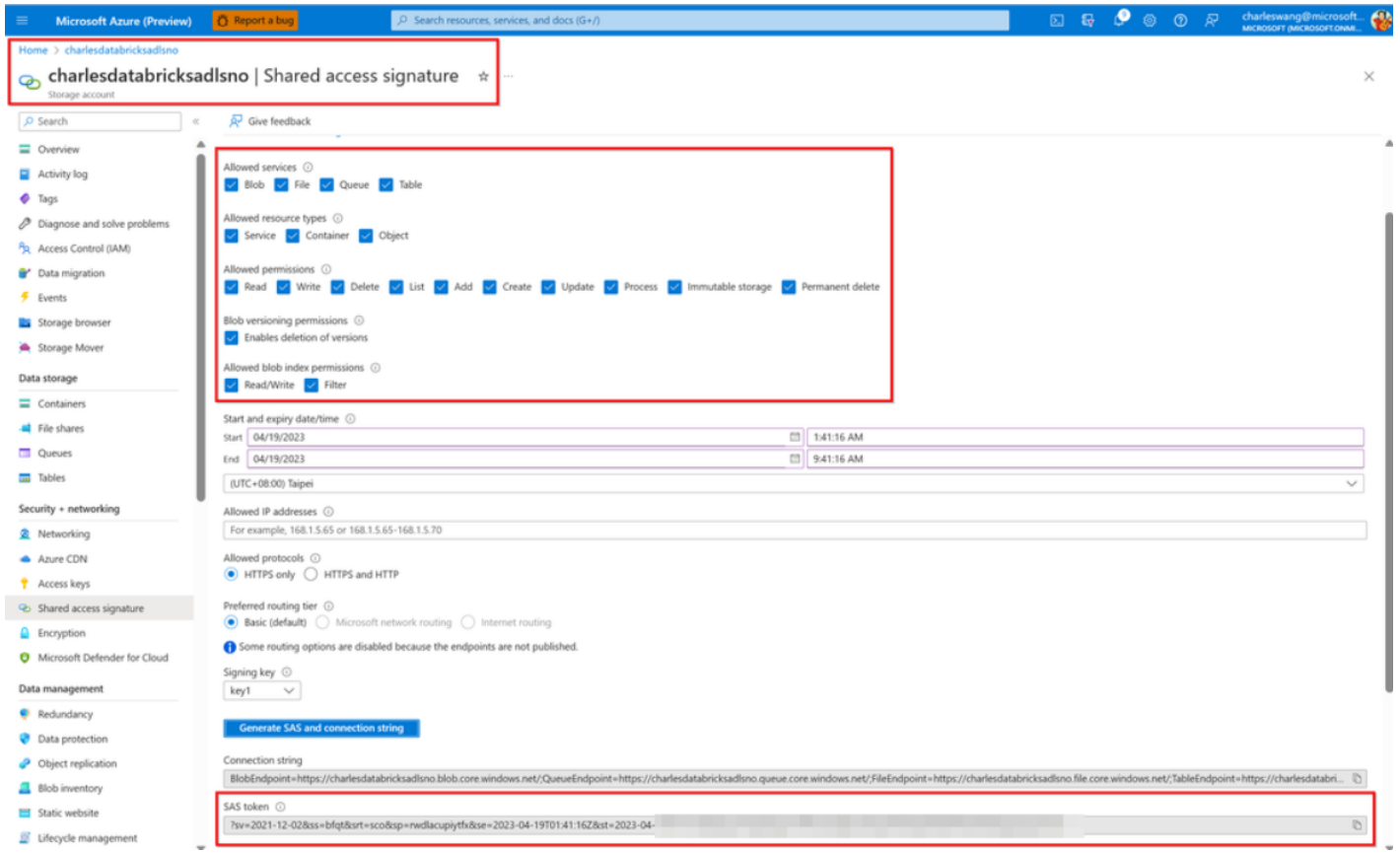
```
/subscriptions/<subscription-id>/resourceGroups/<resource-group>/providers/Microsoft.Databricks/accessConnectors/<connector-name>
```

```
databricks storage-credentials update <my-storage-credential> \
--isolation-mode ISOLATED \
--profile <profile-name>
```

Connect to Azure Data Lake Storage (ADLS) Gen2 and Blob Storage

Access Azure Data Lake Storage Gen2 or Blob Storage with

- Managed identity
- OAuth 2.0 with a Microsoft Entra ID service principal
- Shared access signatures (SAS)
- Account keys



Configure Azure service principal

- Create Azure service principal using *spark.conf.set* in notebooks

```
service_credential = dbutils.secrets.get(scope="<secret-scope>",key="<service-credential-key>")

spark.conf.set("fs.azure.account.auth.type.<storage-account>.dfs.core.windows.net", "OAuth")

spark.conf.set("fs.azure.account.oauth.provider.type.<storage-account>.dfs.core.windows.net",
"org.apache.hadoop.fs.azurebfs.oauth2.ClientCredsTokenProvider")

spark.conf.set("fs.azure.account.oauth2.client.id.<storage-account>.dfs.core.windows.net", "<application-id>")

spark.conf.set("fs.azure.account.oauth2.client.secret.<storage-account>.dfs.core.windows.net", service_credential)

spark.conf.set("fs.azure.account.oauth2.client.endpoint.<storage-account>.dfs.core.windows.net",
"https://login.microsoftonline.com/<directory-id>/oauth2/token")
```

Configure SAS tokens

- Configure SAS tokens for multiple storage accounts in the same Spark session

```
spark.conf.set("fs.azure.account.auth.type.<storage-
account>.dfs.core.windows.net", "SAS")

spark.conf.set("fs.azure.sas.token.provider.type.<storage-
account>.dfs.core.windows.net",
"org.apache.hadoop.fs.azurebfs.sas.FixedSASTokenProvider")

spark.conf.set("fs.azure.sas.fixed.token.<storage-
account>.dfs.core.windows.net",
dbutils.secrets.get(scope="<scope>", key="<sas-token-key>"))
```

Configure Account key

- Databricks recommends using a Microsoft Entra ID service principal or a SAS token to connect to Azure storage instead of account keys

```
spark.conf.set( "fs.azure.account.key.<storage-account>.dfs.core.windows.net",

dbutils.secrets.get(scope="<scope>", key="<storage-account-access-key>"))
```

Access Azure storage

- Once you have properly configured credentials to access your Azure storage container, you can interact with resources in the storage account using URIs

Python

Copy

```
spark.read.load("abfss://<container-name>@<storage-account-name>.dfs.core.windows.net/<path-to-data>")

dbutils.fs.ls("abfss://<container-name>@<storage-account-name>.dfs.core.windows.net/<path-to-data>")
```

SQL

Copy

```
CREATE TABLE <database-name>.<table-name>;

COPY INTO <database-name>.<table-name>
FROM 'abfss://container@storageAccount.dfs.core.windows.net/path/to/folder'
FILEFORMAT = CSV
COPY_OPTIONS ('mergeSchema' = 'true');
```

Connect to cloud object storage and services using Unity Catalog

- Databricks recommends using Unity Catalog to manage access to all data that you have stored in cloud object storage
- Unity Catalog provides a suite of tools to configure secure connections to cloud object storage

```
/subscriptions/<subscription-id>/resourceGroups/<resource-group-name>/providers/Microsoft.ManagedIdentity/userAssignedIdentities/<managed-identity-name>
```

Path-based access to cloud storage

- Unity Catalog supports path-based access to external tables and external volumes using cloud storage URIs
- Databricks recommends that users read and write all Unity Catalog tables using table names and access non-tabular data in volumes using */Volumes* paths

```
/Volumes/<catalog>/<schema>/<volume>/<path>/<file-name>
```

```
dbfs:/Volumes/<catalog>/<schema>/<volume>/<path>/<file-name>
```

Demo

Connect to Azure Data Lake Storage Gen2 and Blob Storage

Leverage Azure Databricks for Azure Cosmos DB Operations

Introduction to Azure Cosmos DB

- A fully managed, globally distributed NoSQL database service designed for high availability, elastic scalability, and low latency performance
- Key features include
  - Multi-model support (document, key-value, graph, and column-family data)
  - Global distribution with multi-region writes
  - Guaranteed single-digit millisecond response times
  - Flexible consistency models

Integration of Azure Databricks and Azure Cosmos DB

- Easily ingest, process, and analyze data from Azure Cosmos DB using Azure Databricks Spark-based analytics engine
- Combine low-latency data access of Azure Cosmos DB with streaming capabilities of Azure Databricks for real-time insights
- Leverage Azure Databricks SQL capabilities to perform complex queries on Azure Cosmos DB data
- Implement a Lakehouse architecture by combining NoSQL capabilities of Azure Cosmos DB with Azure Databricks Delta Lake technology

Integration Patterns

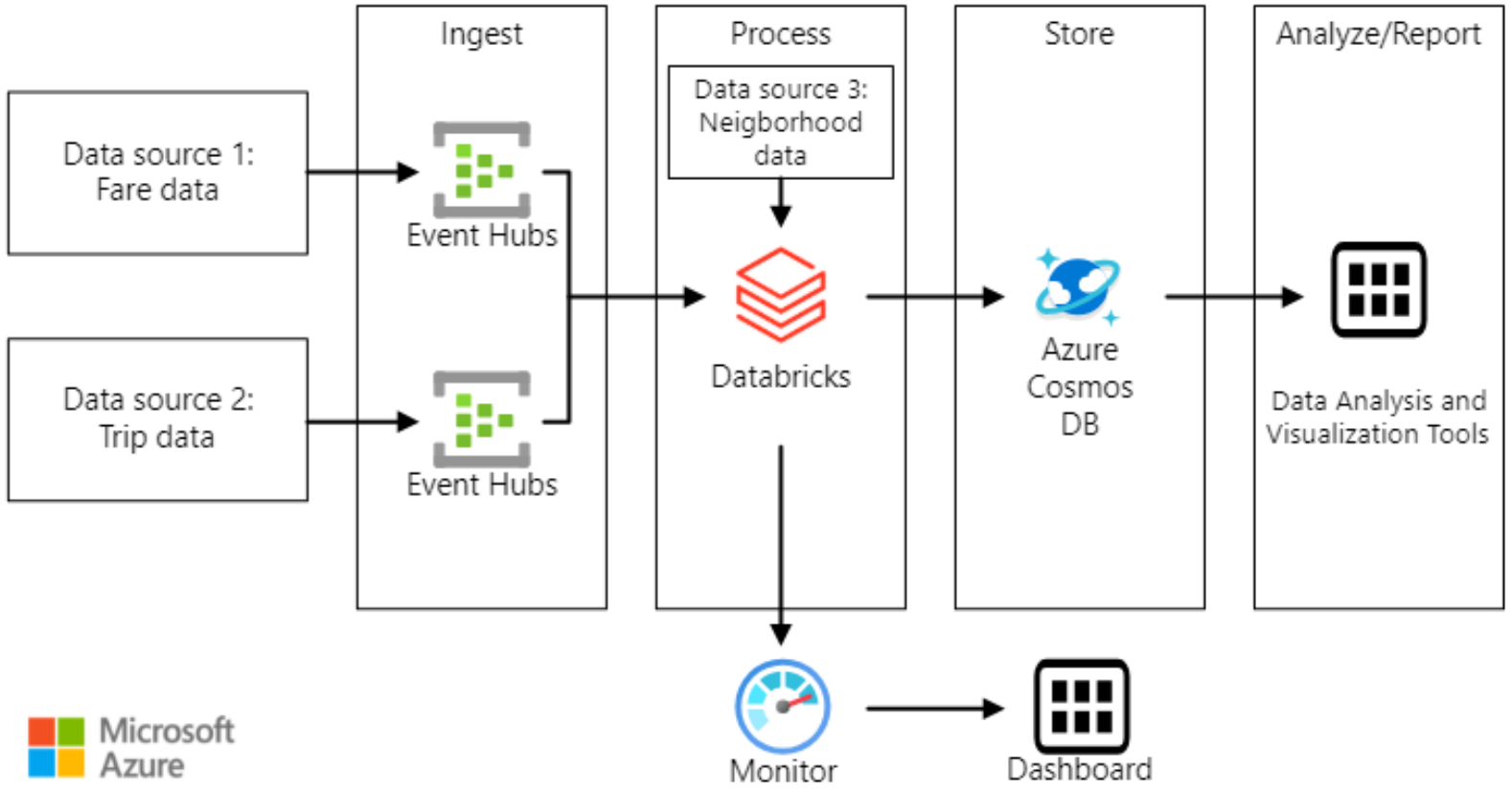
- Batch Read and Write
- Streaming with Change Feed
- Bulk Insert and Upsert
- Analytics and Aggregations

```
#Read the data into a Spark dataframe and print the count
cosmos_df = spark.read.format("cosmos.oltp").options(**readCfg).load()

# Process data
processed_df = cosmos_df.filter(cosmos_df.age > 30).select("id", "name", "age")

# Write back to Cosmos DB
processed_df.write.format("cosmos.oltp").options(**readCfg).mode("append").save()
```

Reference Architecture



Azure Cosmos DB Spark Connector

- Azure Cosmos DB Spark Connector is the foundation of Azure Cosmos DB and Azure Databricks integration
- It enables seamless data transfer and processing between the two services

Connect to an API for NoSQL account by using Spark

- Use your existing Azure Databricks workspace to create a compute cluster

Python

```
# Set configuration settings
config = {
  "spark.cosmos.accountEndpoint": "<nosql-account-endpoint>",
  "spark.cosmos.accountKey": "<nosql-account-key>",
  "spark.cosmos.database": "cosmicworks",
  "spark.cosmos.container": "products"
}
```

Create a database and a container

- Use the Catalog API to manage account resources such as databases and containers
- You can use OLTP to manage data within the container resources

Python Copy

```python
# Create a database by using the Catalog API
spark.sql(f"CREATE DATABASE IF NOT EXISTS cosmosCatalog.cosmicworks;")
```

Ingest data

- Create a sample dataset
- Use OLTP to ingest that data to the API for NoSQL container

Python Copy

```python
# Ingest sample data
spark.createDataFrame(products) \
  .toDF("id", "category", "name", "quantity", "price", "clearance") \
  .write \
  .format("cosmos.oltp") \
  .options(**config) \
  .mode("APPEND") \
  .save()
```

Query data

- Load OLTP data into a data frame to perform common queries on the data
- You can use various syntaxes to filter or query data

Python Copy

```
# Render results of raw query
rawQuery = "SELECT * FROM cosmosCatalog.cosmicworks.products WHERE price > 800"
rawDf = spark.sql(rawQuery)
rawDf.show()
```

Perform common operations

- When you work with API for NoSQL data in Spark, you can perform partial updates or work with data as raw JSON

Python

```
# Create data frame
spark.createDataFrame(patchProducts) \
  .write \
  .format("cosmos.oltp") \
  .options(**configPatch) \
  .mode("APPEND") \
  .save()
```

Demo

Connect to Azure Cosmos DB for NoSQL by using Spark

Secret management with Azure Key Vault

Azure Databricks Secret Management

Use Azure Databricks secrets to store your credentials and reference them in notebooks and jobs

- Create a secret scope
- Add secrets to the scope
- Assign permissions on the secret scope
- Reference secrets in your code

Bash

```
databricks secrets create-scope jdbc
```

Bash

```
databricks secrets put-secret jdbc username
databricks secrets put-secret jdbc password
```

Secret scopes

There are two types of secret scope

- Azure Key Vault-backed
- Azure Databricks-backed

```Bash
databricks secrets create-scope <scope-name>
```

Create an Azure Key Vault-backed secret scope

- Create an Azure Key Vault-backed secret scope using
  - Azure portal
  - Azure Databricks workspace UI
  - Databricks CLI

Python

```
username = dbutils.secrets.get(scope = "jdbc", key = "username")
password = dbutils.secrets.get(scope = "jdbc", key = "password")

df = (spark.read
  .format("jdbc")
  .option("url", "<jdbc-url>")
  .option("dbtable", "<table-name>")
  .option("user", username)
  .option("password", password)
  .load()
)
```

Create a secret in an Azure Key Vault-backed scope

- A secret is a key-value pair that stores sensitive material using a key name that is unique within a secret scope
- Use the Azure portal or Azure Set Secret REST API to create a secret in Azure Key Vault

Python

```python
from databricks.sdk import WorkspaceClient

w = WorkspaceClient()

w.secrets.put_secret("<secret_scope>","<key-name>",string_value ="<secret>")
```

Manage secret scope permissions

The user who creates the secret scopes is granted the MANAGE permission. This allows the scope creator to

- Read secrets in the scope
- Write secrets to the scope
- Manage permissions on the scope

Bash

```
databricks secrets put-acl <scope-name> <principal> <permission>
```

Bash

```
databricks secrets list-acls <scope-name>
```

Demo

Create and use a Databricks secret

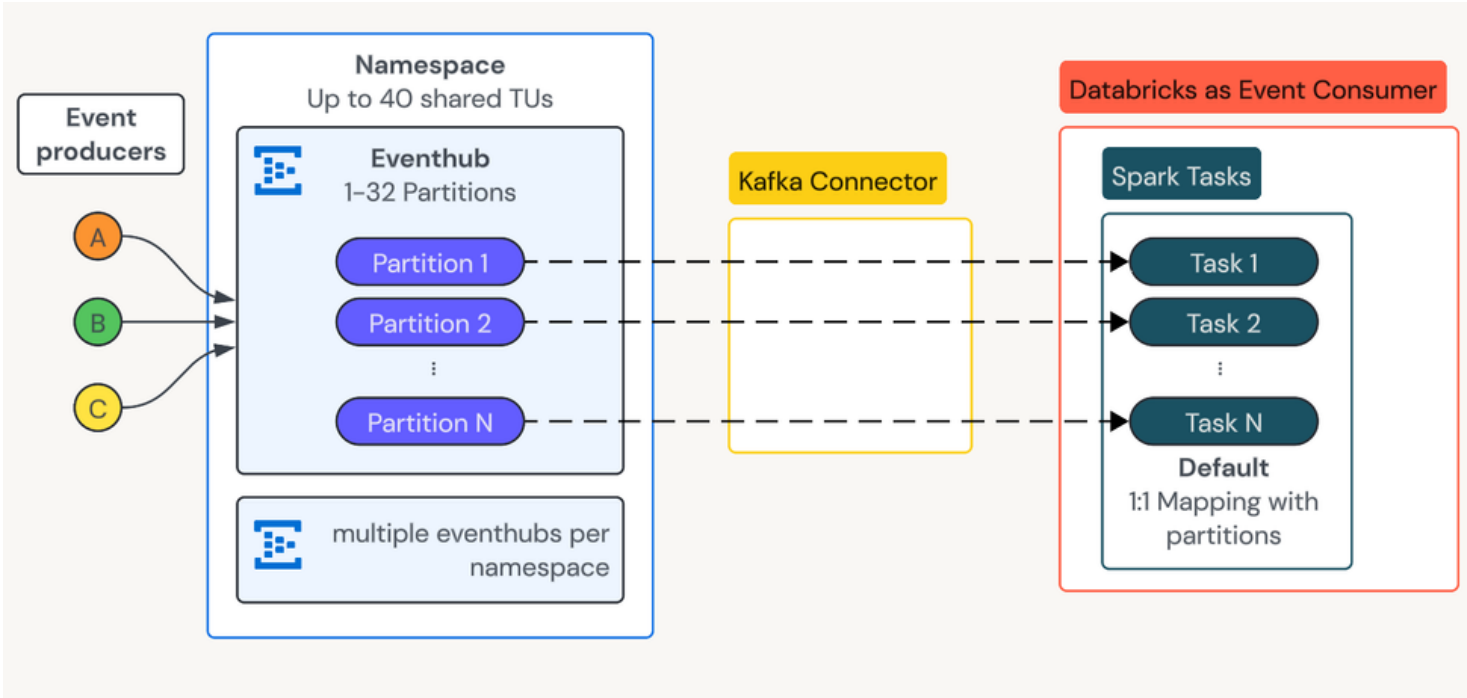
Connect Azure Databricks to Azure Event Hubs

Connect Lakeflow Declarative Pipelines to Azure Event Hubs

- Use Lakeflow Declarative Pipelines to process messages from Azure Event Hubs
- Azure Event Hubs provides an endpoint compatible with Apache Kafka that you can use with the Structured Streaming Kafka connector, available in Databricks Runtime

Steps to connect a Lakeflow Declarative Pipelines to an existing Azure Event Hubs instance

- Store the policy key in an Azure Databricks secret
- Create a notebook and add the pipeline code to consume events
- Create the pipeline



Store the policy key in an Azure Databricks secret

- Use Azure Databricks secrets to store and manage access to the key

```
databricks --profile <profile-name> secrets
create-scope <scope-name>

databricks --profile <profile-name> secrets
put-secret <scope-name> <shared-policy-name> --
string-value <shared-policy-key>
```

Create a notebook and add the pipeline code

- Databricks recommends using the Lakeflow Declarative Pipelines pipeline settings to configure application variables
- The pipeline code then uses the *spark.conf.get()* function to retrieve values

```
@dlt.create_table(
     comment="Raw IOT Events",
     table_properties={
        "quality": "bronze",
        "pipelines.reset.allowed": "false"
     },
     partition_cols=["eh_enqueued_date"] )
@dlt.expect("valid_topic", "topic IS NOT NULL")
@dlt.expect("valid records", "parsed_records IS NOT NULL")
def iot_raw():
return (
     spark.readStream
        .format("kafka")
        .options(**KAFKA_OPTIONS)
        .load()
        .transform(parse)
)
```

Create the pipeline

- Create a new pipeline
- It uses an Azure Data Lake Storage Gen2 (ADLS Gen2) storage account

```
{
"clusters": [
    { "spark_conf": {
       "spark.hadoop.fs.azure.account.key.<storage-account-name>.dfs.core.windows.net":
            "{{secrets/<scope-name>/<secret-name>}}" }, "num_workers": 4
    }
],
"development": true,
"continuous": false,
"channel": "CURRENT",
"edition": "ADVANCED",
"photon": false,
"libraries": [
    { "notebook": {
       "path": "<path-to-notebook>"
       }
    }
],
"name": "dlt_eventhub_ingestion_using_kafka",
"storage": "abfss://<container-name>@<storage-account-name>.dfs.core.windows.net/iot/",
"configuration": { "iot.ingestion.eh.namespace": "<eh-namespace>",
"iot.ingestion.eh.accessKeyName": "<eh-policy-name>",
"iot.ingestion.eh.name": "<eventhub>",
"io.ingestion.eh.secretsScopeName": "<secret-scope-name>",
"iot.ingestion.spark.maxOffsetsPerTrigger": "50000",
"iot.ingestion.spark.startingOffsets": "latest",
"iot.ingestion.spark.failOnDataLoss": "false",
"iot.ingestion.kafka.requestTimeout": "60000",
"iot.ingestion.kafka.sessionTimeout": "30000"
   },
   "target": "<target-database-name>"
}
```

Coming up next...

Day 1

Module 1 - Introduction to Azure Databricks
- Azure Databricks: A Data Intelligent Platform
- Why Azure Databricks
- Decision guide: Azure Databricks vs. Microsoft Fabric

Module 2 - Migration to Azure Databricks
- Microsoft Cloud Adoption Framework for Azure
- Migration strategies
- Data landing zones
- Migration scenarios

Interactive Simulated Lab Experience
- End-to-End Streaming Pipeline with Lakeflow Declarative Pipelines in Azure Databricks

Day 2

Module 3 - Integration with Azure
- Seamless integration with Microsoft Azure services
- Connect to Azure Data Lake Storage (ADLS) Gen2 and Blob Storage
- Leverage Azure Databricks for Azure Cosmos DB Operations
- Secret management with Azure Key Vault
- Connect Azure Databricks to Azure Event Hubs

Module 4 - Integration with Microsoft Fabric and Power BI
- Data Intelligence with Azure Databricks and Microsoft Fabric
- Connect Power BI to Azure Databricks
- Integration with Azure Data Factory
- Mirroring Azure Databricks Unity Catalog

Interactive Simulated Lab Experience
- Setup and use Unity Catalog for Data Management in Azure Databricks
- Real-Time Streaming with Azure Databricks and Azure Event Hubs

Day 3

Module 5 - Integration with Azure AI Foundry
- Azure Databricks connector in Azure AI Foundry
- Mosaic AI and machine learning on Azure Databricks
- Query Generative AI model serving endpoints
- Databricks Assistant, AI/BI Genie and AI Functions on Azure Databricks
- Chat with LLMs and prototype GenAI apps using AI Playground
- Build and optimize agents on your data with Agent Bricks

Module 6 - Security and Governance
- Integrate Azure Databricks with Microsoft Purview
- Integration of Azure Databricks Unity Catalog with Microsoft Purview

Module 7 - Well-architected for Azure Databricks
- Lakehouse implementation: Principles and best practices
- Azure Databricks well-architected framework

Interactive Simulated Lab Experience
- Responsible AI with Large Language Models using Azure Databricks and Azure OpenAI
- Connect to and manage Azure Databricks in Microsoft Purview

Microsoft

Technofocus

Thank You!