**Databricks, Cloud Platforms, and Data Engineering Evaluation**

**1. Databricks Usage**

**Question:** Explain how Databricks handles big data processing and analytics. How would you use Databricks to implement a machine learning workflow?

Databricks handles big data processing and analytics by combining the power of Apache Spark with a user-friendly platform. It enables one to process massive datasets quickly using distributed computing and supports multiple languages, including Python, SQL, Scala, and R. With its built-in Delta Lake, it ensures data reliability and performance through ACID transactions and optimizations such as caching and Z-ordering.

To implement a machine learning workflow in Databricks, I typically follow these steps:

Ingest and clean data using Spark or Delta Lake.

Explore and visualize the data with notebooks (Python, SQL, or built-in visual tools).

Feature engineering with PySpark or Pandas API.

Train models using MLlib, scikit-learn, or other ML libraries.

Track experiments using MLflow (built-in).

Deploy models as batch jobs or real-time endpoints.

Monitor and retrain as needed using Databricks Jobs and Model Registry**.**

**2. Unity Catalog Migration**

**Question:** What are the key considerations and steps for migrating a data catalog to Databricks Unity Catalog? How does Unity Catalog improve data governance?

**Key Considerations Before Migration**

1. Catalog Structure Planning: Design your *metastore* → *catalogs* → *schemas* → *tables/views* hierarchy.
2. Access Control Mapping: Convert existing ACLs to Unity Catalog’s fine-grained RBAC model.
3. Data Lineage Impact: Ensure tools support Unity Catalog lineage (e.g., Power BI, dbt).
4. Delta Compatibility: Only *Delta tables* are supported for managed access. Parquet/CSV must be converted.
5. Workspaces Link: One metastore per region; link multiple workspaces as needed.

Unity Catalog improves data governance by centralizing access control, enabling fine-grained permissions, enforcing consistent security policies across workspaces, and providing built-in auditing and data lineage.

**3. ETL Pipeline Development**

**Question:** Describe how you would design and optimize an ETL pipeline to process large-scale datasets in Azure or AWS. Include details on tools and technologies you would use.

To design and optimize an ETL pipeline for large-scale datasets in Azure, I’d use **Azure Data Factory** for orchestration, **Azure Databricks** for scalable transformations on Delta Lake, and **ADLS Gen2** for storage—ensuring partitioning, caching, and parallel processing are in place, while applying data quality checks and logging to keep things clean, fast, and reliable.

In one of the projects in my current company - optum, I built a Databricks pipeline that ingested 10M+ claim records daily from ADLS using Auto Loader, transformed the data with Delta Live Tables, and orchestrated end-to-end workflows via ADF—achieving 40% faster load times and improved data quality validations.

**4. Cloud Platform Proficiency**

**Question:** Compare Azure Blob Storage and AWS S3 for data storage. What are the advantages and limitations of each, and how would you decide which to use?

Azure Blob Storage and AWS S3 are both scalable object storage solutions. S3 offers broader third-party integration, superior cross-region replication, and mature ecosystem tools. Azure Blob integrates deeply with Azure services, provides tiered access, and supports native Data Lake features. S3 excels in global scalability, while Blob is ideal for hybrid Azure environments. Choice depends on ecosystem alignment, latency needs, cost models, and regulatory compliance across cloud-native workloads.

**5. Orchestration Tools**

**Question:** How would you implement and automate a data pipeline using Azure Data Factory or Apache Airflow? Provide an example use case.

To implement and automate a data pipeline in **Azure Data Factory (ADF)**, I would create linked services to connect source (e.g., SQL Server) and sink (e.g., Azure Data Lake), define datasets, and build a pipeline with copy activities, data flow transformations, and triggers for scheduling. I’d use parameterized pipelines for reusability and monitor executions via ADF’s UI or alerts.

**Example:** I once worked in a project, where I automated daily ingestion of on-prem SQL transactional data into ADLS, transformed it using ADF data flows, and loaded curated outputs into Synapse for reporting—reducing manual intervention and enabling near real-time financial insights.

**6. Data Modeling and Optimization**

**Question:** How do you approach schema design and database optimization for analytics workloads? Provide a specific example from your experience.

For analytics workloads, I design schemas using star or snowflake models, ensuring dimension tables are normalized and fact tables denormalized for efficient joins. I optimize queries with clustered column store indexes, partitioning, and materialized views. In my last project, I modeled claims data in Synapse using a star schema, partitioned fact tables by service date, and indexed key dimensions—improving Power BI report performance by and reducing storage costs through compression and intelligent tiering.

**7. Programming and Problem-Solving**

**Question:** Compare two implementations of calculating average user spending from a transaction dataset. Explain the key architectural differences between PySpark and Python approaches. How would each perform with 1 million transactions? What about 1 billion?

In Python (e.g., using pandas), average user spending is calculated via df.groupby('user\_id')['amount'].mean(), suitable for small to medium datasets in-memory. PySpark uses df.groupBy("user\_id").agg(avg("amount")), designed for distributed processing. With 1 million records, both perform well—Python is simpler, PySpark adds overhead. With 1 billion records, Python struggles with memory limits and performance, while PySpark distributes computation across clusters, ensuring scalability. Architecturally, Python is single-node, in-memory; PySpark is multi-node, lazy-evaluated, and fault-tolerant—ideal for big data pipelines and production workloads in cloud environments like Databricks or EMR.

**8. Data Governance Policies**

**Question:** What steps do you take to enforce data governance policies in a large-scale data processing environment?  
  
To enforce data governance in large-scale processing, I implement role-based access control (RBAC), data classification, encryption (at rest/in transit), auditing, and data lineage tracking. I use tools like Unity Catalog (Databricks) or Azure Purview for metadata management and policy enforcement. I also define data retention rules and PII masking strategies.

In a current project, I used Unity Catalog to restrict access to PHI columns and enabled audit logging for all Delta table access in Databricks.Top of Form

Bottom of Form

**9. Troubleshooting**

**Question:** Describe a time you encountered a significant issue with a data pipeline. How did you identify the problem, and what was your solution?

During a nightly Delta Live Tables pipeline, load latency ballooned from ten minutes to over two hours, blocking downstream forecasting dashboards. I first checked pipeline run logs and observed huge shuffle spill warnings. Spark UI revealed a single skewed fact-table partition containing 80% of records. Profiling the raw data showed a new upstream process appending the same customer\_id for all failed payments, creating imbalance. I mitigated immediately by adding a conditional salt key and repartitioning with salting\_key, customer\_id, then instituted an Auto Loader expectation to quarantine malformed rows. Latency returned to norma

**10. Emerging Technologies**

**Question:** What recent trends or technologies in data engineering are you most excited about, and how do you see them impacting the role of a Data Engineer in the near future?

I'm most excited about the rise of lakehouse architectures, data mesh decentralization, and real-time streaming with Apache Flink/Kafka, combined with MLOps automation and generative AI for SQL/code generation. These trends shift Data Engineers toward platform-building, governance, and enabling self-service analytics, making the role more strategic and collaborative.