Databricks Data Intelligence Platform

1. The Lakehouse Paradigm

Databricks is built on the **Lakehouse Architecture**. Historically, data was split between **Data Lakes** (cheap, unstructured storage) and **Data Warehouses** (structured, high-performance SQL databases). The Lakehouse unifies these, providing the cost-efficiency of a lake with the performance and ACID transactions of a warehouse.

2. Core Architecture: The Planes

Databricks operates using a "separation of concerns" across three planes:

- **Control Plane:** The web UI where you write code in notebooks and manage workflows.
- **Compute Plane:** The actual Spark clusters (VMs) that execute your code. These scale horizontally.
- **Data Plane:** Your cloud storage (AWS S3 / Azure Data Lake) where the raw files reside.

3. Key Technical Features

A. Collaborative Notebooks

Unlike standard Jupyter, Databricks Notebooks support:

- **Multi-language Cells:** Use `python`, `sql`, `scala` in the same file.
- **Real-time Pairing:** Multiple engineers can code in the same cell simultaneously.
- **Git Integration:** Native "Repos" allow you to sync with GitHub for version control.

B. Delta Lake (The Storage Engine)

Delta Lake is the open-source storage layer that brings reliability to your B.Tech projects:

- **ACID Transactions:** Ensures your data doesn't get corrupted during failed writes.
- **Time Travel:** Query older versions of data using `VERSION AS OF`.
- **Schema Enforcement:** Prevents "garbage in" by checking data types during ingestion.

C. Unity Catalog (Governance)

Unity Catalog is the centralized security layer. It manages:

- **Access Control:** Granting permissions to tables, files, and models.
- **Data Lineage:** A visual graph showing how data moved from a raw CSV to a final ML prediction.

4. AI & Machine Learning Features (Mosaic AI)

Databricks handles the "MLOps" lifecycle through **Mosaic AI**:

Feature	Purpose for AI/ML Students
MLflow	Tracks experiments, logs hyperparameters, and versions your models.
Feature Store	A central repository to store and reuse "features" across different models.
Model Serving	Deploys your model as a REST API endpoint with one click.
Vector Search	Essential for building RAG (Retrieval-Augmented Generation) applications.

---

5. Hands-on Workflow (Sample Tutorial)

Step 1: Create a Cluster

Navigate to **Compute** > **Create Cluster**. Select "Personal Compute" for small university projects. Ensure "Autoscaling" is on to save costs.

Step 2: Ingest Data

Use **Auto Loader** to incrementally ingest files.

Python
```python
# Python snippet for your notebook
df = (spark.readStream
    .format("cloudFiles")
    .option("cloudFiles.format", "csv")
    .load("/databricks-datasets/samples/population-data"))
```

Step 3: Transform with Delta

SQL
```sql
-- SQL snippet to create a Delta table
CREATE TABLE population_silver
USING DELTA
AS SELECT * FROM population_raw_temp;
```

Step 4: Track an ML Experiment

Python

```
import mlflow
with mlflow.start_run():
    model = train_model(data)
    mlflow.log_metric("accuracy", 0.95)
    mlflow.log_model(model, "my_ai_model")
```

6. Comparison Table: Why Use Databricks

Aspect	**Traditional Big Data**	**Databricks**
Management	Manual Hadoop/Spark setup	Fully Managed / Serverless
Governance	Fragmented (IAM roles)	Centralized (Unity Catalog)
ML Lifecycle	Hard to track and deploy	Native (MLflow + Model Serving)