16 Databricks Interview Questions from basic to advanced concepts

Basic Databricks Concepts Questions

What is Databricks, and how does it differ from traditional Spark clusters?

"**Databricks** is essentially a simplified, cloud-based platform built around **Apache Spark**. Think of it as Spark that's been packaged up and given a lot of extra tools.

The difference from a **traditional Spark cluster** is that Databricks handles almost all the messy parts for you—like installing Spark, managing infrastructure, security, and setting up the cluster. With traditional Spark, you have to do all of that yourself on a set of machines (VMs). Databricks is the **easy button** for running big data analytics."

Explain the concept of a Databricks workspace.

"The **Databricks workspace** is your shared online home for all your work. It's like a shared folder or a team project portal.

It's a web-based environment where you can create **notebooks**, manage your libraries, share code with teammates, schedule jobs, and access data. Everything you do on Databricks happens inside a workspace."

What are notebooks in Databricks, and how are they used?

"**Notebooks** are the main place where you write and run code. They mix code (like Python or SQL) with text, explanations, and visualizations.

They're used for everything from **data exploration** and writing quick scripts to building full-scale **ETL pipelines**. They are great because you can see your results immediately, and they make it easy to document and share your analysis steps."

Describe the difference between Databricks jobs and interactive clusters.

"The difference comes down to purpose and cost:

- **Interactive Clusters** (or *All-Purpose* clusters) are for **development and exploration**. You start them up, attach your notebooks, run code, and experiment. You pay for them while they're running, even if you're just sitting there thinking.
- **Databricks Jobs** are for **production and scheduled tasks**. You define a notebook or script and tell Databricks to run it at a specific time, and the platform automatically

creates a brand-new cluster just for that task, runs the job, and then *terminates* the cluster. This is much more **cost-effective** for production work."

What is the role of the Databricks File System (DBFS)?

"**DBFS** is a virtual file system that sits on top of your cloud storage (like S3 or Azure Data Lake). Its main role is to give Spark and your notebooks a **simple, familiar place to access data** without having to deal with the complex cloud storage paths every time.

It's essentially a convenience layer. When you use a `/dbfs/` path, Databricks translates that to the actual location in your cloud storage."

Intermediate Databricks Concepts Questions

Explain how Databricks integrates with cloud storage (e.g., AWS S3, Azure Data Lake, GCP GCS).

"Databricks integrates by acting as the **compute layer** that reads and writes data to your cloud storage account. It doesn't actually store the large data files itself—your data lives in the cloud (like in an Azure Data Lake).

The key is that you tell the cluster your **credentials** (keys, service principals, etc.) so it has permission to access those storage accounts. Once connected, Databricks uses Spark to process the files directly where they live, making the storage practically unlimited."

What is the difference between Delta Lake and traditional data lakes?

"A **traditional data lake** is just a large pile of files (like Parquet or CSV) in cloud storage. The problem is they lack structure, often have bad data, and you can't easily update data in place.

Delta Lake is an **open-source storage layer** on top of that file storage that adds structure and reliability. It turns your pile of files into a reliable database by adding features like:

1. **ACID Transactions** (reliability).
2. **Schema Enforcement** (data quality).
3. The ability to **update/delete data** easily."

How does Databricks handle cluster auto-scaling and job scheduling?

"Databricks handles these automatically to save time and money:

- **Auto-scaling:** When you start a cluster, you set a minimum and maximum number of worker nodes. If a Spark job needs more power (like when processing a massive data set), Databricks will **automatically add** nodes up to your maximum. When the job is done, it will **automatically remove** the extra nodes.
- **Job Scheduling:** This is handled through the **Jobs** feature in the workspace. You can use a simple UI to set up a regular schedule (e.g., 'run this notebook every night at 2 AM'). Databricks' own infrastructure manages the timing and cluster provisioning."

Explain Databricks' role in ETL/ELT pipelines.

"Databricks is primarily the **"T" (Transformation) engine** in the pipeline.

- In an **ETL** model, it *Transforms* the data using its powerful Spark compute, and then *Loads* it into a target system like a data warehouse.
- In an **ELT** model (which is more common now), it *Loads* the raw data into the data lake, and then uses Spark/Delta Lake to *Transform* that data *in place* before it's used for analysis.

It offers tools like **Delta Live Tables (DLT)** to make building, monitoring, and maintaining these pipelines much simpler with declarative code."

What are Databricks' different cluster modes (Standard, High Concurrency, Single Node), and when would you use each?

The modes are defined by how many users can access them:

- **Single Node:** Only useful for very small data processing or testing where you don't need Spark's parallelism. It has only one machine.
- **Standard:** Best for **single users** or small teams where only one person or one job is accessing the cluster at a time. It's the default for interactive use.
- **High Concurrency:** Designed for environments where **multiple users** might simultaneously be running queries or where you need to share a cluster between several jobs. It uses stronger isolation features to ensure one user's query doesn't crash another's.

Advanced Databricks Concepts Questions

How does Delta Lake ensure ACID transactions on big data?

"Delta Lake ensures **ACID** (Atomicity, Consistency, Isolation, Durability) by using a **Transaction Log**.

1. When you run an operation (like an update or an insert), it doesn't modify the data files directly.
2. Instead, it writes the changes as a small **JSON file** to the transaction log, and new Parquet files are created for the data.
3. The transaction log acts as the single source of truth. Only when the log file is successfully written is the transaction considered committed.
4. This log allows it to instantly know which files are the most current version, ensuring no user sees partially written data (Atomicity/Isolation) and guaranteeing data integrity."

Describe how Databricks implements streaming pipelines using Structured Streaming.

"Databricks uses **Apache Spark's Structured Streaming** library. It treats a continuous stream of data (from Kafka, Kinesis, etc.) as an **ever-growing table**.

You write the streaming pipeline using standard Spark DataFrame operations (the same code you'd use for batch processing!), and Spark handles the complexity. Databricks makes it simple to deploy this, often using **Delta Live Tables (DLT)**. DLT lets you define your pipeline stages (e.g., Raw to Silver to Gold tables) using declarative SQL or Python, and it automatically manages the continuous operation, error handling, and recovery for you."

What are Z-ordering and data skipping in Databricks, and how do they improve performance?

"Both are features that speed up queries on Delta Lake:

- **Data Skipping:** Delta Lake automatically collects **metadata** (like minimum and maximum values) about the data in each file. When you run a query, Spark checks this metadata and **skips over entire data files** that couldn't possibly contain the data you're looking for, saving a ton of time.
- **Z-ordering:** This is a technique where you physically **colocate related information** in the same set of files. For example, if you often filter by `customer_id` and `product_category`, Z-ordering sorts the files so that a specific combination of those values is grouped together, dramatically improving the effectiveness of data skipping and making queries run much faster."

How would you optimize a slow-performing Databricks job (tuning clusters, caching, partitioning)?

"I would use a systematic approach, starting with the simplest fixes:

1. **Examine the Spark UI:** I'd first look at the execution plan to see where the time is spent—is it on **shuffling** (moving data between stages) or I/O?
2. **Data Optimization:**

    - Ensure the data is stored in **Delta Lake**.
    - Run `OPTIMIZE` on the Delta table to compact small files.
    - Check for relevant **Z-ordering** on frequently filtered columns.
3. **Cluster Tuning:**
    - Is the cluster sized appropriately for the workload? Is there enough memory to avoid spilling to disk?
    - Tune the **parallelism** (e.g., setting the number of partitions) to ensure all cores are utilized but not overloaded.
4. **Code/Logic Optimization:**
    - **Caching** is a temporary fix, but helpful for iterative jobs (like ML training).
    - Push down predicates by filtering as early as possible (`WHERE` clause) to reduce the data volume that Spark has to process.
    - Use highly optimized functions like **`window functions`** instead of custom UDFs."

Explain how Databricks Unity Catalog helps with governance and security in multi-tenant environments.

"**Unity Catalog** is Databricks' central point for **data and AI governance**. In a multi-tenant (shared) environment, its role is to create a single, clear, and consistent set of rules:

- **Centralized Security:** It allows you to define permissions (who can read/write what) on data objects (**tables, views, etc.**) using familiar **SQL commands**, just like a traditional database. This access applies consistently across all workspaces, languages (SQL, Python, Scala), and compute resources.
- **Metadata Management:** It acts as a single source of truth for all your data, providing consistent **schema and lineage** (where data came from and where it goes) across the entire cloud account.
- **Multi-Cloud Ready:** It's designed to manage data that might be spread across multiple cloud accounts, ensuring your governance rules are enforced everywhere."

How would you design a cost-optimized Databricks architecture for a production workload?

"The main goal is to minimize wasted cluster-on time:

1. **Use Databricks Jobs (Automated Clusters):** Never run production jobs on interactive clusters. Set up the job to use a **new, dedicated cluster** that **terminates immediately** after the job finishes.
2. **Smart Auto-scaling:** Set the cluster's minimum nodes to a low number (or zero) and a sensible maximum. Use **auto-stop** to turn off the cluster if it sits idle for a short time (e.g., 5-10 minutes).
3. **Spot/Preemptible Instances:** For non-critical workloads, use **Spot instances** (AWS/GCP) or **Low-priority VMs** (Azure) for the worker nodes. Databricks can handle

the loss of these nodes and automatically replace them, saving up to 70% on compute cost.

4. **Optimize the Code:** The fastest, cheapest job is the one that processes the least amount of data. This means aggressive **filtering, Z-ordering, and using Delta Lake** features to skip data."