

Getting Started with Databricks

Traditional data management has relied on two primary paradigms: DWH and DL. Data lakes, while flexible, often struggle with data quality and governance due to their unstructured nature. Data warehouses, though structured, can be rigid and costly, limiting their adaptability to the evolving demands of diverse, high-volume, and high-velocity data. To overcome these challenges, enterprises used to go with two-tier architecture, DL for storing raw data for AI applications and DWH for BI. This strategy has increased complexity, increase in ETL jobs, and complicated Data governance. Databricks addresses this issues by offering a unified platform that supports both data lake and data ware house called "data lake house".

Understanding the Databricks Platform

The platform is built on Apache Spark. A data lakehouse represents a hybrid solution that combines the best aspects of data lakes and data warehouses. Specifically, it integrates the openness, scalability, and cost efficiency of data lakes with the reliability, strong governance, and performance features of data warehouses.

Consider an example of managing books,

DWH is like a library where books are all sorted for efficient analysis and querying. However, maintaining this system is costly, and its rigid structure makes it challenging to accommodate new or unconventional book formats.

DL is like a vast, inexpensive, and unstructured repo where all books are hoarded without proper organization or processing. It's like a big shelf where items are just hoarded. Locating a specific item can be problematic.

A lakehouse represents a smart, adaptable library that combines the best of both worlds: the vast, flexible, and economic storage of a data lake with the structured, organized, and analyzable system of a data warehouse.

These enables organizations to store diverse data of vast amounts in a low-cloud storage while maintaining the ability to analyze it efficiently and securely. This facilitates various tasks such as data engineering, ML and analytics in one place. Thus, Data LakeHouse (DLH) serves as a unified platform where DE, DS, ML, etc. can all work together.

High Level Architecture of Databricks Lakehouse

The Databricks Lakehouse is designed with a layered architecture that consists of four fundamental layers: the cloud infrastructure, Databricks Runtime, data governance, and the workspace. Each of these layers plays a pivotal role in ensuring the platform's scalability, reliability and security.

Cloud infrastructure:

Foundational layer. Databricks is a multi-cloud platform. This layer is responsible for providing the underlying hardware resources that Databricks accesses on behalf of users. It enables provisioning of storage, networking, and the VMs or nodes that form the backbone of a computing cluster running Databricks runtime.

Databricks Runtime:

It's a pre-configured VM image optimized for use within Databricks clusters. It includes a set of components, such as Apache Spark, Delta Lake, and other essential libraries. Delta Lake enhances traditional DL's by providing ACID transactional guarantees, thereby ensuring improved data reliability and consistency.

Data governance with Unity Catalog:

The core of the Databricks LH architecture. It provides centralized data governance solution across all data and AI assets. Unity Catalog is designed to secure and manage data access across the Databricks env, ensuring access only to authorized users. This layer is crucial for governance, security, integrity and compliance across the lakehouse platform.

Databricks workspace:

Top of the architecture, it serves as the UI for interacting with the platform. Here users of different personas can perform their respective operations. It provides an interactive env for different personas, it also offers range of services such as notebooks, viz dashboards, workflow management tools for orchestration.

Deployment of Databricks Resources

When deploying Databricks resources within your cloud provider's environment, the architecture is divided into two high-level components: the control plane and the data plane.

Control Plane:

This is managed by Databricks and hosts various platform services within the Databricks account. When workspace is created it is deployed within the control plane, along with other essential services such as the Databricks UI, cluster manager, workflow service, and notebooks. Thus, CP manages workspace management, cluster provisioning, and job scheduling. It also provides the UI to interact with the platform, including web-notebooks, Databricks REST-API, and the CLI.

Data Plane:

It resides within the user's own cloud subscription. This is where the actual storage and compute resources (non-serverless) are provisioned and managed. When a user sets up a Spark cluster, VM that comprise the cluster are deployed in the data plane, within the user's cloud account. Also, storage resources, such as those used by Databricks File System (DBFS) or Unity Catalog, are also deployed in the data plane.

This separation of control and data planes offers several advantages. First, it ensures that the compute and storage resources remain within the user's cloud environment, providing

greater control over data security and compliance. Second, it allows Databricks to manage the operational aspects of the platform, such as updates and maintenance, without impacting the user's data or compute resources.

Apache Spark on Databricks

Apache Spark an OS data processing engine, is a cornerstone of the Databricks platform, enabling fast and scalable analytics.

Key features of Apache Spark on Databricks includes:

Distributed data processing:

Spark's architecture is designed to process data in parallel. On Databricks, this capability is enhanced by the seamless integration with cloud-based clusters, which can be scaled up or down depending on the workload.

In-memory processing:

One of the main advantages of Spark is its in-memory processing capability. By keeping data in memory across the cluster, Spark significantly reduces time required for iterative algos and complex computations.

Databricks supports all languages that Spark does: Python, Java, Scala, R, SQL.

Batch and Stream Process:

It supports both batch and stream processing.

Flexible data handling:

Databricks, powered by Spark, can handle structured, semi-structured, and unstructured data.

Databricks File System (DBFS)

The DBFS acts as an abstraction layer that simplifies file management across the distributed environment. It allows users to interact with cloud files as if they were stored on a local file system. When a file is created in a Databricks cluster and stored in the DBFS, it is actually persisted in the underlying cloud storage associated with your cloud provider.

Ex: File stored in DBFS on Azure Databricks would be stored in ADLS. This design ensures that data remains durable and accessible, even after Spark cluster is terminated.

Exploring the Databricks Workspace

The layout is intuitively organized into two primary sections: the sidebar and the top bar.

Sidebar: Located left side of the interface, offers quick access to the platform's key services.

Workspace: Integrated browser where resources such as folders, notebooks, and other files are organized and managed.

Catalog: Allows you to manage your data and AI assets, such as DB's, ML models, etc.

Workflows: Place to deploy and orchestrate jobs, for automated processing and execution of your data tasks.

Compute: Tab to create and manage compute resources such as classic clusters and pools.

SQL, DE and ML personas

Top bar:

Search bar: AI-powered search tool to search for various items within workspace.

Switch Workspaces: To navigate between different projects.

Databricks Assistant: AI-based workspace assistant designed to enhance your experience with developing notebooks, queries, and dashboards.

Profile Settings: User related ops such as managing preferences, linking external services, and notifications. Admin-specific settings to config workspace env.

Workspace Navigation:

Home Directory: Default location within the workspace. Its personalized to each user's personal directory, providing semi-private space to store your files and folders.

Workspace Directory: Root folder containing all users' personal directories.

Repos: Legacy service to integrate with Git repos, now replaced by "Git Folders"

Trash: Recycle bin, after 30 days files are removed.

Creating Clusters

Clusters form the backbone of data processing and analytics on the platform. A cluster is a collection of computers, nodes, or VMs acting as a single entity. In the context, of Spark cluster comprises of a master node aka "driver node" and several worker nodes. Driver node is responsible for orchestrating the activities of the worker nodes, which execute tasks in parallel.

Databricks offers two primary types of clusters: all-purpose clusters and job clusters. Each serves distinct purposes and use cases, tailored to different stages of the data engineering and analytics lifecycle.

	All Purpose Cluster	Job Cluster
Usage	Interactive development and data analysis	Automated job execution

	All Purpose Cluster	Job Cluster
Management	Manually created and managed by the user	Automatically created by the job scheduler
Termination	Manual or auto-termination after inactivity	Automatic termination upon task completion
Cost efficiency	Comes at a higher expense	Less expensive

All Purpose Cluster:

Usage

All-purpose clusters are mainly used for interactive tasks where a user is actively involved. This includes writing and testing code in notebooks and performing exploratory data analysis (EDA). The interactive nature of these clusters makes them essential for development and testing environments.

Management

Users can manually create and manage their all-purpose clusters depending on their needs. This can be achieved using the Databricks workspace interface, command-line interface, or REST API.

Termination

All-purpose clusters can be terminated manually by the user when they are no longer needed. Additionally, Databricks provides an auto-termination feature, where you can specify a period of inactivity after which the cluster will automatically shut down. This feature is particularly useful in reducing costs, as it prevents unnecessary resource consumption when the cluster is idle.

Cost efficiency

All-purpose clusters cost more to run when compared to other types of clusters. Additionally, they can become even more expensive due to the need for manual control and termination. Although auto-termination helps with cost savings, it still enforces a minimum runtime of 10 minutes, which can add to the overall expense.

Job Clusters: These clusters are designed to be ephemeral, spinning up only when a job is triggered and terminating immediately after the job is completed:

Usage

Job clusters are used primarily for running automated tasks, such as scheduled jobs and data pipelines. They are particularly useful in production environments where tasks need to be executed without manual intervention. Examples include extract, transform, and load

(ETL) jobs, database maintenance, and training machine learning models on a scheduled basis.

Management

Unlike all-purpose clusters, job clusters are not created manually by the user. Instead, they are automatically provisioned by the Databricks job scheduler when a job is triggered. This automation simplifies cluster management in production, as there is no need for manual intervention to start or stop clusters.

Termination

Job clusters are designed to be used for a single purpose and terminate automatically once the assigned task is completed. This ephemeral nature ensures that resources are utilized only when necessary, which helps in optimizing costs and enhancing the efficiency of resource allocation.

Cost efficiency

From a cost-efficiency standpoint, job clusters are generally more economical than all-purpose clusters. Therefore, it is recommended to use job clusters for production environments to optimize costs.

Databricks Pools

In addition to offering various types of clusters, Databricks provides cluster pools to further optimize resource usage and reduce operational latency. Cluster pools are a powerful tool for users who need to minimize the time it takes to spin up clusters, especially in environments where job execution speed is critical.

A cluster pool in Databricks is essentially a group of pre-configured, idle virtual machines that are ready to be assigned to clusters as needed. The primary advantage of using a cluster pool is the reduction in both cluster start time and autoscaling time whenever there are available nodes in the pool. This can be particularly beneficial in scenarios where time is a critical factor, such as in automated report generation and real-time data processing tasks.

Although, Databricks itself doesn't charge for the idle instances in a pool, cloud provider does. Because, these instances are actively running on your cloud infra and hence they incur standard compute costs. Therefore, when using cluster pools, it is important to balance the need for rapid cluster availability with the associated cloud costs.

Choose Databricks Runtime 13.3 LTS. Photon is an optional feature to enhance cluster's performance. Its a high-speed query engine developed in c++, designed to accelerate SQL queries in Spark. Costs could be increased.

A DBU (databricks unit) is a measure of processing capacity per hour, which helps estimate the costs associated with running the cluster.

Magic Commands: Special cell instructions that provide additional functionality in the notebook env. Prefixed with a %, allows you to execute tasks that go beyond standard code execution.

%sql --> Changes the cell language to sql

%md --> Markdown for documenting analysis, adding notes, or structuring notebooks into sections.

%run --> Allows you to execute another notebook within the current notebook.

%fs --> Simple way to execute file system ops directly within your notebook cells. It allows you to copy, move and delete files and directories within the cloud storage.

Databricks Utilities

dbutils provides a range of utility commands for interacting with different services and tools within Databricks, including file system. (dbutils.fs)

```
files = dbutils.fs.ls("/databricks-datasets")
```

same as %fs ls /databricks-datasets, but it stores the output.

If you need to perform a quick, one-off file system operation, the %fs magic command is straightforward and easy to use. For more complex tasks, especially when you need to manipulate the output programmatically, dbutils is the better choice. It allows you to store the results in variables, apply conditional logic, loop through files, and more—all within your Python code.