Chapter 1. Getting Started with Databricks

Databricks is transforming the way data and artificial intelligence (AI) are managed with its innovative Data Intelligence Platform. This platform offers a unified solution that addresses the limitations of traditional data systems, providing a more comprehensive approach to work with data. In this chapter, we will explore the Databricks Data Intelligence Platform and its capabilities. We will begin with an overview of the platform's architecture and then delve into its key features, including compute resource creation, notebook execution, and Git integration.

Introducing the Databricks Platform

Traditional data management has long relied on two primary paradigms: data lakes and data warehouses. Each approach comes with its own strengths and limitations, particularly in the context of big data processing. 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 often deploy multiple systems—data lakes for storing raw data for AI applications, and data warehouses for business intelligence (BI) purposes. However, this strategy leads to increased complexity, requires frequent data transfers, and complicates data governance. Databricks addresses these issues by offering a unified platform that supports both data lake and data warehouse functionalities in a single environment, known as the *data lakehouse*.

Understanding the Databricks Platform

The Databricks Data Intelligence Platform is an AI-powered data lakehouse platform 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. To illustrate the concept of a data lakehouse, imagine you have a vast collection of books (data) that contains various genres, formats, and authors. Traditionally, two separate systems would be employed to manage these books:

Data warehouse

An organized library where books (data) are carefully curated, processed, and stored in a specific format, facilitating efficient analysis and querying. However, maintaining this system is costly, and its rigid structure makes it challenging to accommodate new or unconventional book formats.

Data lake

A vast, inexpensive, and unstructured repository where all the books are stored without extensive organization or processing. It resembles an endless, disordered storage shelf where various items can be easily stored; however, 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.

In the real world, such integration ensures that enterprises can store vast amounts of diverse data in low-cost cloud storage while maintaining the ability to analyze it efficiently and securely. This facilitates performing a wide variety of tasks in one place, including data engineering, machine learning, and analytics. Thus, the data lakehouse serves as a unified platform where data engineers, data scientists, and analysts can all work together. Figure 1-1 illustrates this convergence of capabilities into a single, comprehensive platform.

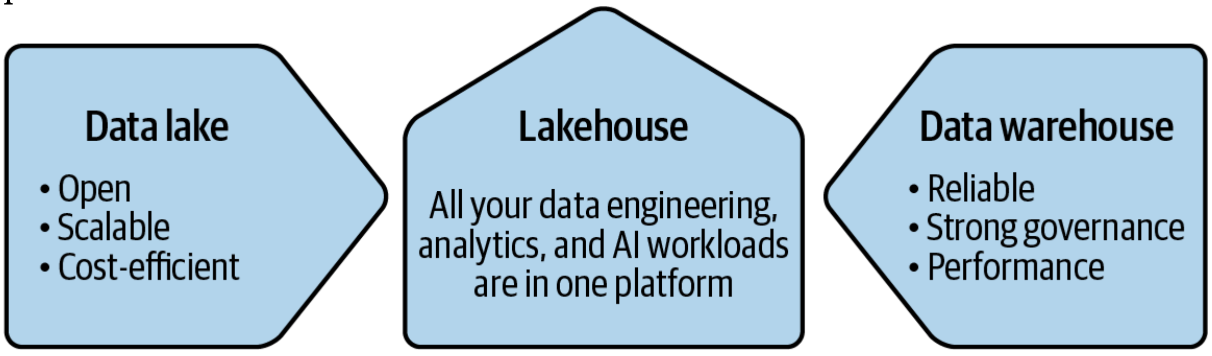


Figure 1-1. Convergence of data lakes and data warehouses into a unified data lakehouse platform

To understand how Databricks achieves this, let's examine the underlying architecture of its data lakehouse.

High-Level Architecture of the 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. Figure 1-2 illustrates the high-level architecture of the Databricks lakehouse, showcasing the relationships among these layers.

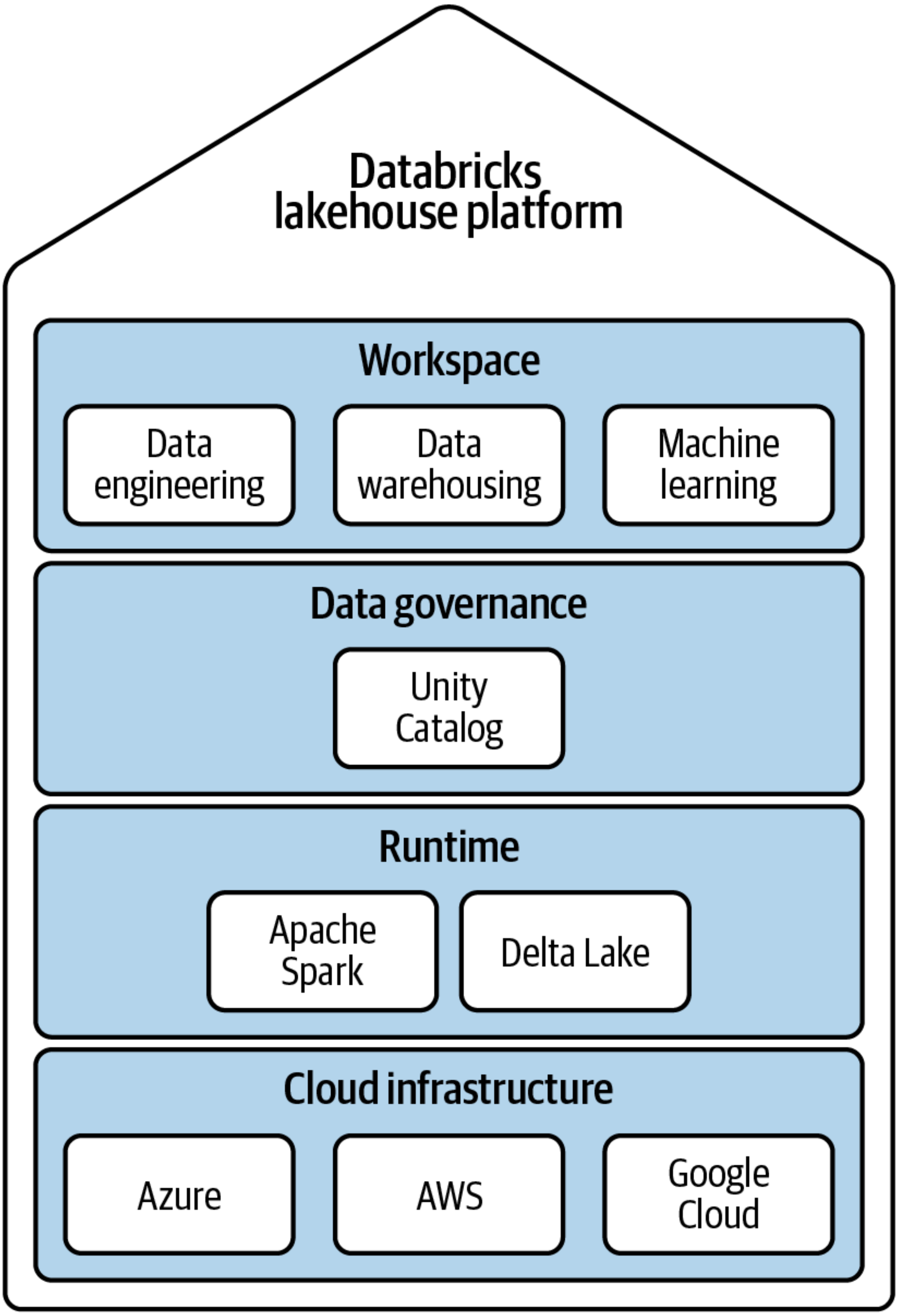


Figure 1-2. High-level architecture of the Databricks lakehouse

Each of these layers plays a vital role in ensuring the platform's scalability, reliability, and security. To gain a deeper understanding of their individual contributions, let's examine each layer in detail, starting from the bottom:

Cloud infrastructure
: At the foundation of the Databricks lakehouse architecture lies the cloud infrastructure layer. Databricks is a multi-cloud platform, meaning it is available on major cloud service providers, including Microsoft Azure, Amazon Web Services (AWS), and Google Cloud Platform (GCP). This layer is responsible for providing the underlying hardware resources that Databricks accesses on behalf of users. It enables the provisioning of essential components, such as storage, networking, and the virtual machines (VMs) or nodes that form the backbone of a computing cluster running Databricks Runtime.

Databricks Runtime
: Databricks Runtime is a pre-configured virtual machine image optimized for use within Databricks clusters. It includes a set of core components, such as Apache Spark, Delta Lake, and other essential system libraries. Delta Lake enhances traditional data lakes by providing transactional guarantees similar to those found in operational databases, thereby ensuring improved data reliability and consistency. In Chapter 2, we explore Delta Lake in detail to understand its transformative impact on data lake reliability.

Data governance with Unity Catalog
: At the core of the Databricks lakehouse architecture is Unity Catalog, which provides a centralized data governance solution across all data and AI assets. Unity Catalog is designed to secure and manage data access across the Databricks environment, ensuring that sensitive information is accessible only to authorized users. This layer is crucial for maintaining data security, integrity, and compliance across the lakehouse platform. Chapter 8 provides an in-depth look at Unity Catalog and its comprehensive features and capabilities.

Databricks workspace
: At the top of the architecture is the Databricks workspace, which serves as the user interface for interacting with the platform. It provides an interactive environment where users can perform data engineering, analytics, and AI workloads using a variety of languages, such as Python, SQL, R, and Scala. The workspace offers a range of services, including notebooks for development, dashboards for visualizing data, and workflow management tools for orchestrating data pipelines.

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. Figure 1-3 illustrates these two components and the interaction between them.

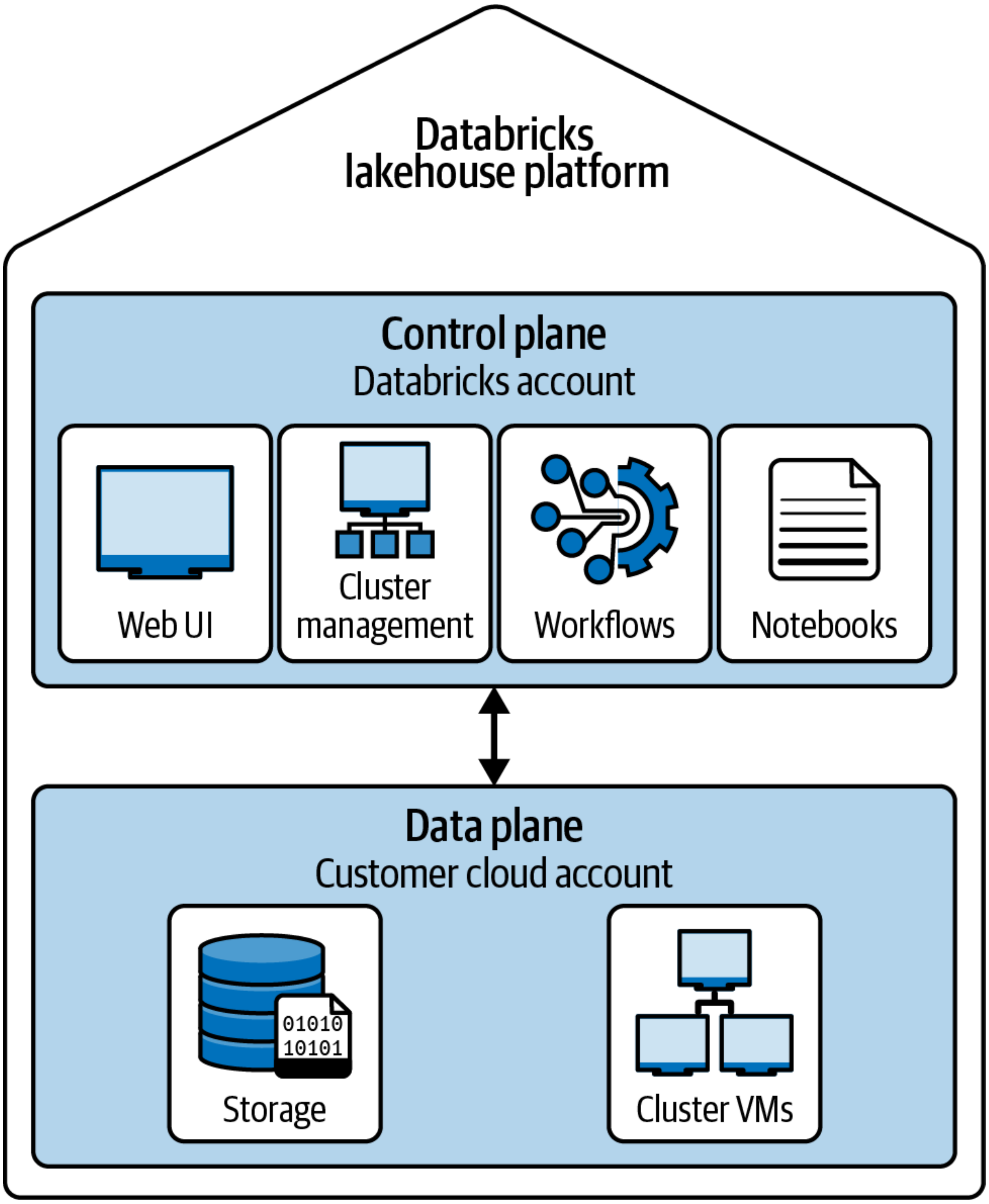


Figure 1-3. Databricks resource deployment architecture

Understanding the distinction between these components is essential for effectively managing and securing your Databricks environment. Let's take a closer look at each component to ensure a clear understanding of their individual roles:

Control plane

The control plane is managed by Databricks and hosts various platform services within the Databricks account. When you create a Databricks workspace, it is deployed within the control plane, along with essential services such as the Databricks user interface (UI), cluster manager, workflow service, and notebooks. Thus, the control plane handles tasks such as workspace management, cluster provisioning, and job scheduling. It also provides the interface through which users interact with the platform, including the web-based notebooks, the Databricks REST API, and the command-line interface (CLI).

Data plane

The data plane, on the other hand, resides within the user's own cloud subscription. This is where actual storage and classic compute resources (non-serverless) are provisioned and managed. When a user sets up a Spark cluster, the virtual machines that comprise the cluster are deployed in the data plane, within the user's cloud account. Similarly, storage resources, such as those used by the 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 open source data processing engine, is a cornerstone of the Databricks platform, enabling fast and scalable analytics. Databricks, founded by the original creators of Apache Spark, has deeply integrated Spark into its platform, making it one of the most optimized environments for running Spark applications.

The key features of Apache Spark on Databricks include the following:

Distributed data processing

Spark's architecture is designed to process data in parallel across multiple nodes in a cluster. 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 Spark’s most significant advantages is its in-memory processing capability. By keeping data in memory across the cluster, Spark significantly reduces the time required for iterative algorithms and complex computations.

Multi-language support

Databricks supports all the programming languages that Spark does, including Scala, Python, SQL, R, and Java.

Batch and stream processing

Apache Spark on Databricks supports both batch and stream processing, making it suitable for a variety of use cases. Batch processing is ideal for historical data analysis and data transformations, while stream processing enables real-time analytics and processing of continuous data streams.

Flexible data handling

Databricks, powered by Spark, can handle structured, semi-structured, and unstructured data. This flexibility is crucial in modern data ecosystems where data comes in various forms, such as CSV files, JSON objects, images, videos, and even complex nested data structures.

Databricks File System (DBFS)

A key feature that enhances Spark’s distributed processing capabilities on Databricks is the 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. This is illustrated in Figure 1-4.

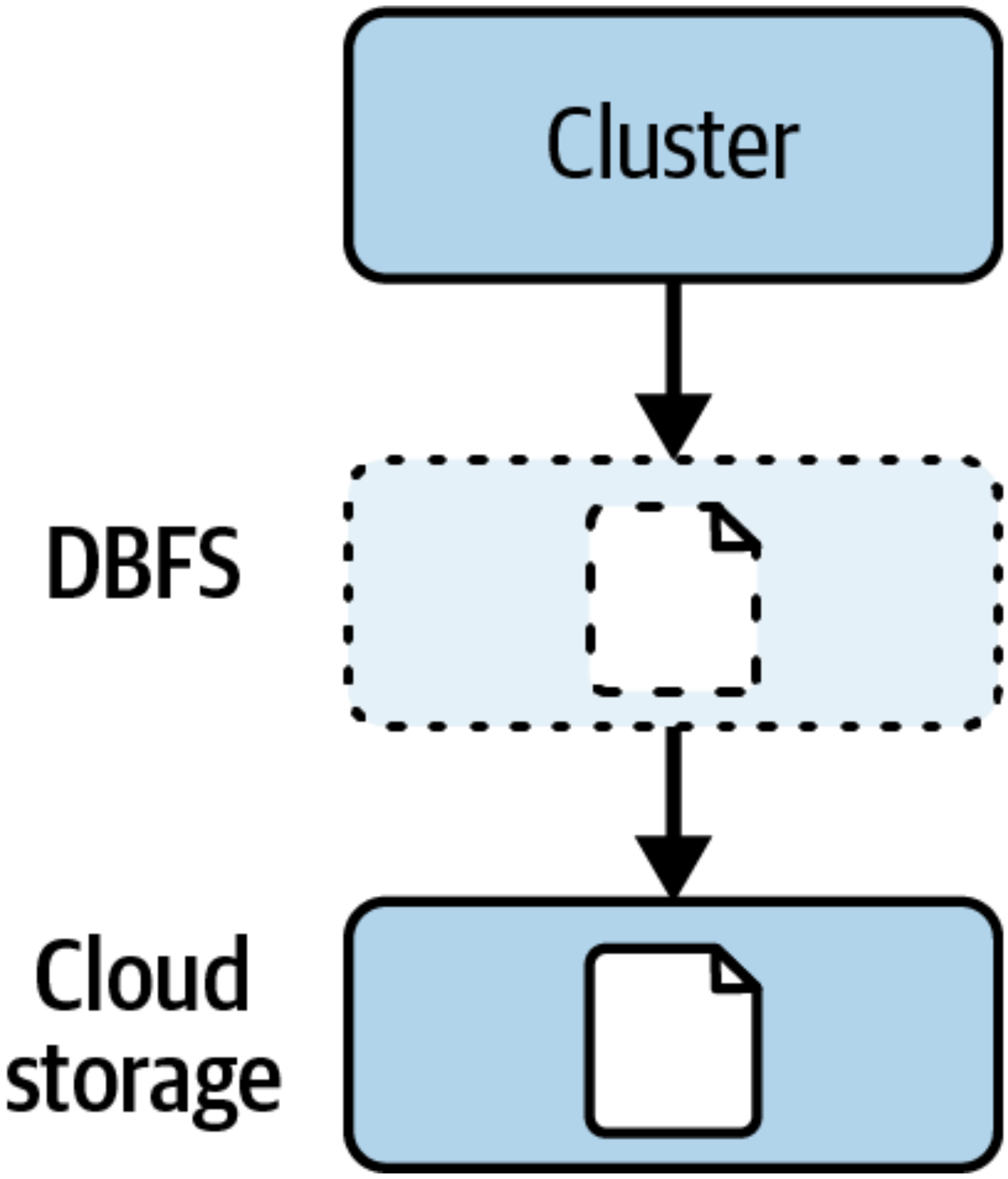


Figure 1-4. Data persistence in the DBFS and the underlying cloud storage

For instance, a file stored in the DBFS on Azure Databricks would really be stored in Azure Data Lake Storage (ADLS). This design ensures that data remains durable and accessible, even after the Spark cluster is terminated.

Setting Up a Databricks Workspace

Creating a Databricks workspace is the first step toward leveraging the platform's capabilities for data engineering, analytics, and machine learning. To set up a workspace, an active Databricks account is required. Databricks offers a 14-day free trial, allowing you to explore its features using your cloud account on Azure, AWS, or Google Cloud. The setup instructions vary slightly depending on the cloud provider you choose; however, it's important to note that the certification exam does not include cloud-specific questions. This means that you won't be tested on the details of creating a workspace on a specific cloud platform.

For detailed instructions on setting up a Databricks workspace on each of these cloud platforms, refer to Appendix A. This section provides step-by-step guidance on how to sign up for a free trial with Databricks and create your first workspace. If you do not have a cloud account or prefer a simpler environment for personal use or training, Databricks also offers the Community Edition. This is a lightweight version of Databricks, which provides access to the key platform's features at no cost. To learn how to sign up for Databricks Community Edition, refer to Appendix B.

Exploring the Databricks Workspace

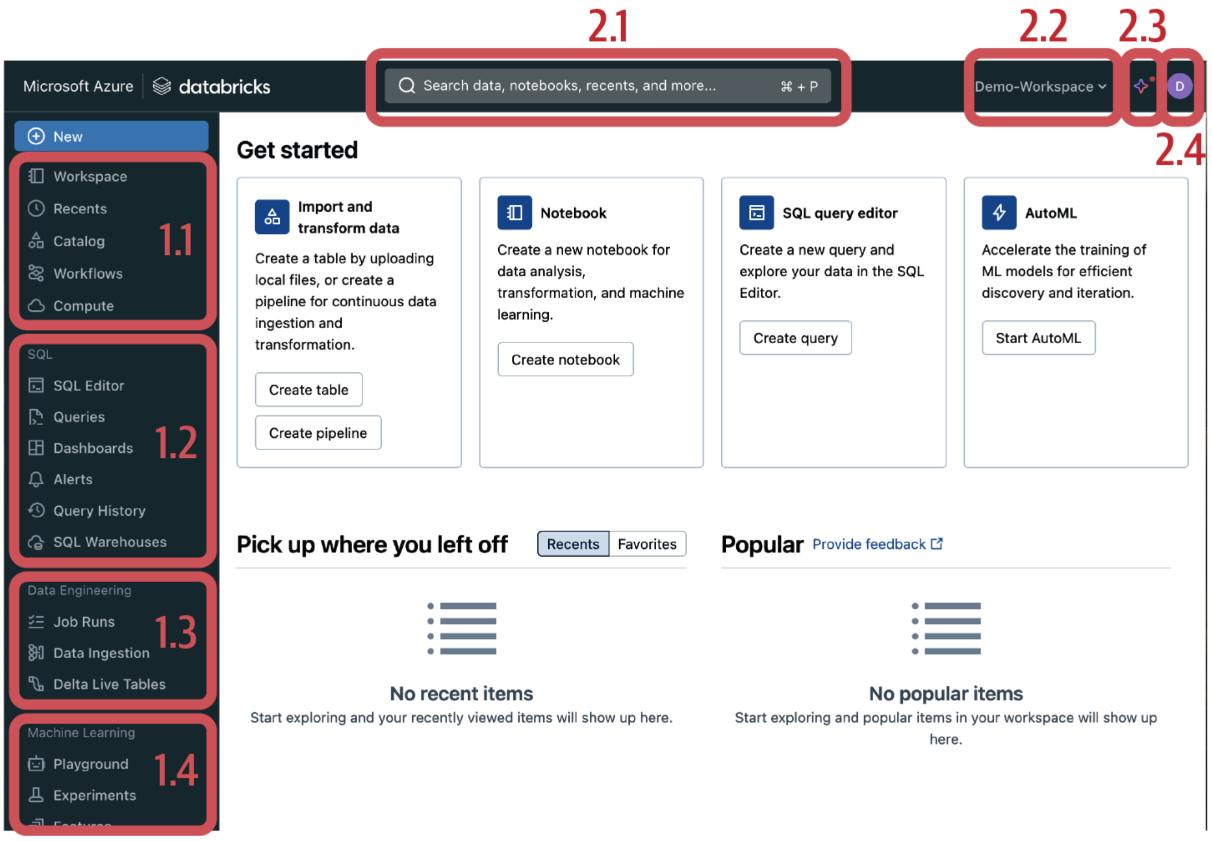
The Databricks workspace provides an easy-to-use and intuitive interface, enabling users to interact with their data objects and perform a wide variety of essential tasks. This represents a unified working environment for data engineers, data analysts, and machine learning engineers.

Overview of the Workspace Interface

Figure 1-5 illustrates the home page of the workspace interface, highlighting several key platform navigation areas.

NOTE

The Databricks platform is frequently updated with enhancements and new features. As a result, the workspace interface in newer versions may differ from the examples provided here. While the core functionality remains the same, the appearance and specific layout elements might vary.



Create a table by uploading local files, or create a pipeline for continuous data ingestion and transformation.

Create a new notebook for data analysis, transformation, and machine learning.

Create a new query and explore your data in the SQL Editor.

Accelerate the training of ML models for efficient discovery and iteration.

Start exploring and your recently viewed items will show up here.

Start exploring and popular items in your workspace will show up here.

Figure 1-5. The home page of Databricks workspace

The interface displays a dynamic landing screen that shows recently accessed items and suggested content, providing a personalized experience. The layout is intuitively organized into two primary sections: the sidebar and the top bar. To break it down further, let’s explore the key components of each section.

Sidebar

The sidebar, located on the left side of the interface, offers quick access to the platform’s key services. It is organized into several categories, each serving a specific function:

Common categories

Workspace

This is an integrated browser where you can organize and manage all your resources, such as folders, notebooks, and other files.

Catalog

This tab allows you to manage your data and AI assets, such as databases, tables, and machine learning models.

Workflows

Here, you can deploy and orchestrate jobs, allowing for automated processing and execution of your data tasks.

Compute
: This tab is where you create and manage your compute resources, such as classic clusters and pools. We'll cover cluster management in detail in the following section.

SQL
: The SQL section provides access to Databricks SQL, a service designed for running SQL workloads on your data. It is particularly useful for analytics and reporting tasks. Chapter 7 provides an in-depth look at Databricks SQL and its capabilities. However, it's important to note that Databricks SQL is not available in the Community Edition. This is one of the reasons why it is recommended to use the full trial in your cloud environment instead of the Community Edition.

Data Engineering
: This section focuses on collaboration among data engineers for performing advanced data engineering tasks. It includes tools and features that are essential for ingesting data and creating data pipelines and jobs. In Chapter 6, we delve into these topics to learn how to build production-grade pipelines and orchestrate jobs effectively.

Machine Learning
: This section offers a range of options tailored for machine learning (ML) engineers. It includes features such as ML experiments, feature stores, and capabilities for registering and serving ML models. It's worth noting that these topics are not included in the Data Engineer Associate certification exam.

Top bar

The top bar spans across the top of the workspace interface and provides several important functions:

Search bar
: This AI-powered search tool allows you to efficiently search for various items within your workspace, including tables, notebooks, dashboards, and more. It is an essential feature for quickly locating resources in your workspace using natural language.

Switch Workspaces
: If you manage multiple workspaces or need to navigate between different projects, the Switch Workspaces option allows you to easily toggle between them.

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

provides a conversational interface that facilitates code generation, explanation, and troubleshooting, thereby boosting your productivity inside the platform. It also integrates with Unity Catalog to offer features such as table searching with context awareness.

Profile settings

The profile settings give you access to user-related options, such as managing your preferences, linking external services, and setting up notifications. They also provide access to admin-specific settings that help configure your workspace environment.

Navigating the Workspace Browser

The workspace browser is a central feature of the Databricks platform, providing an organized and structured environment where you can manage all your project items, such as folders, notebooks, scripts, or other files.

When you navigate to the Workspace tab from the left sidebar, you enter your Home directory, where all your resources are stored, as illustrated in Figure 1-6.

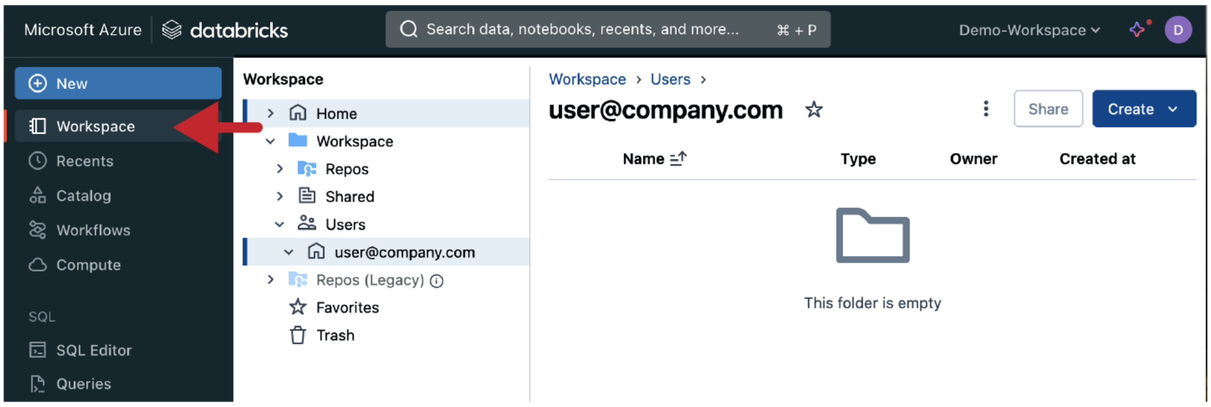


Figure 1-6. The workspace browser in the Databricks platform

The workspace is structured hierarchically, making it easy to organize your work. The left-hand menu includes several key directories to help you manage your workspace effectively, such as the following:

Home directory

The Home directory is your default location within the workspace. It is personalized to each user's personal directory, providing a semi-private space where you can store your files and folders.

Workspace directory

This is the root folder that contains all users' personal directories. From here, you can also access your Home directory by going to *Users > yourname@example.com*.

Repos

This is the legacy service used for integrating your workspace with Git repositories. It has now been replaced by Git folders, which we cover in detail at the end of this chapter in “Creating Git Folders”.

Trash

This folder contains deleted items, which are retained for 30 days before being permanently removed.

To add a new item in your Home directory, click the Create button on the right side of the workspace browser. This allows you to create various types of resources like folders and notebooks, as shown in Figure 1-7.

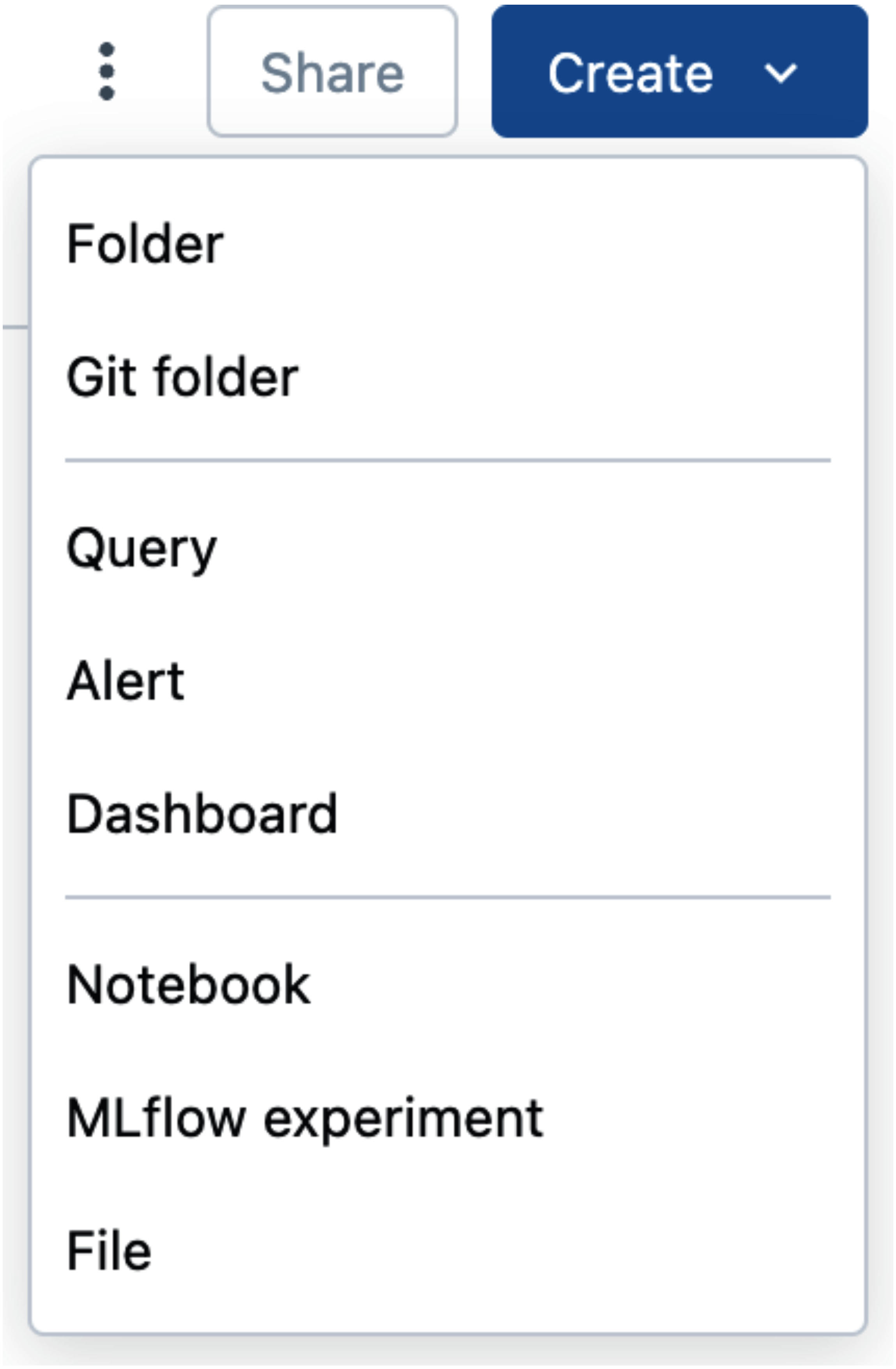


Figure 1-7. Create button in the workspace browser

As an example, use the Create button to add a new folder named *Demo*; then open the folder to begin organizing your files. Within any folder, you can further organize your resources by creating subfolders, which helps to keep your workspace clean and well-organized.
Next to the Create button, you will notice a menu icon represented by three vertical dots, as displayed in Figure 1-8.

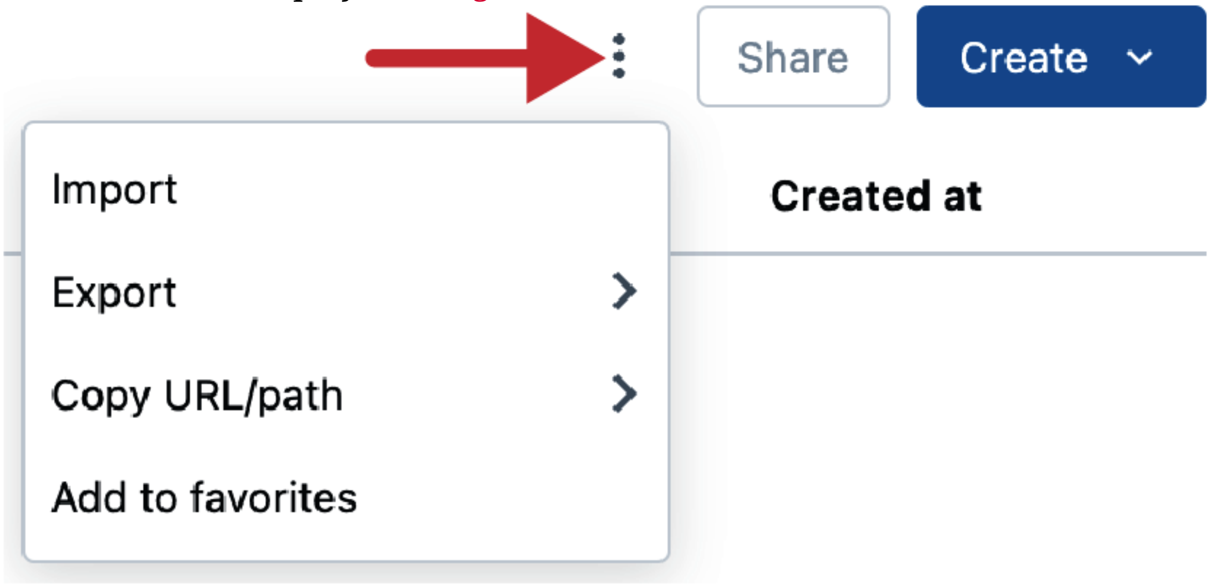


Figure 1-8. Menu icon in the workspace browser

This menu provides additional options for managing your resources, such as importing code files from your local system into your Databricks workspace and exporting the contents of the current folder as an archived file.

Importing Book Materials

The exercises and examples provided in this book are hosted on a GitHub repository. Importing these materials into your Databricks workspace is an essential step to being able to follow along with the content of the book. This section will guide you through the process of importing these resources using two primary methods: Git folders and DBC (Databricks Cloud) files.

Option 1: Git folders

For those using the full version of Databricks on a cloud platform, such as AWS, Azure, or Google Cloud, the Git folders feature offers a seamless integration with Git providers. This allows you to clone remote repositories directly into your Databricks workspace. To clone our book repository from GitHub, follow these steps:

1. Navigate to your workspace browser: In your Databricks workspace, navigate to the Workspace tab to access your Home directory.

2. Create a Git folder: At the top of your directory, click the Create button and select "Git folder" from the drop-down menu, as illustrated in Figure 1-9.

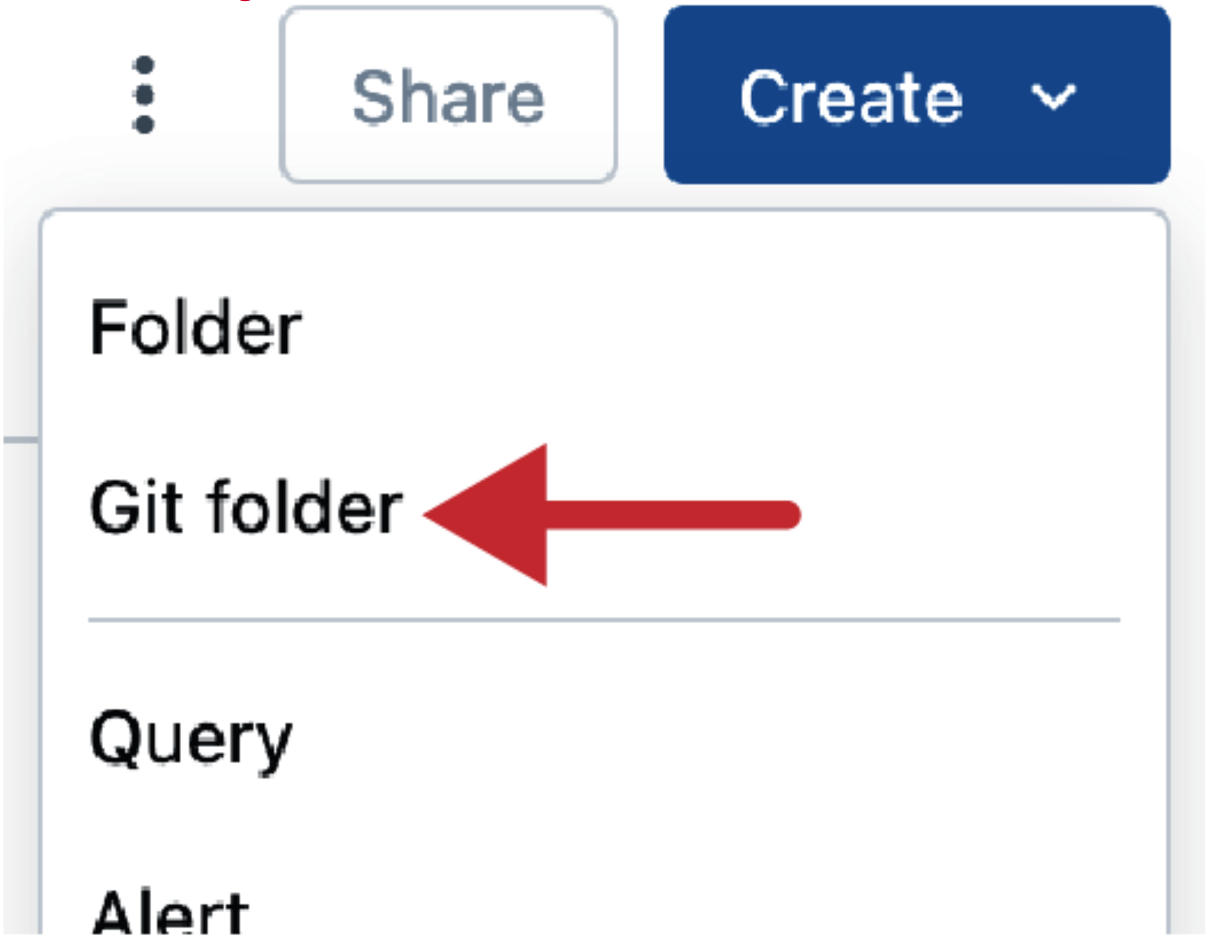


Figure 1-9. Adding a Git folder using the Create button in the workspace browser

This action will open a dialog box where you can specify the GitHub repository you want to clone, as shown in Figure 1-10.

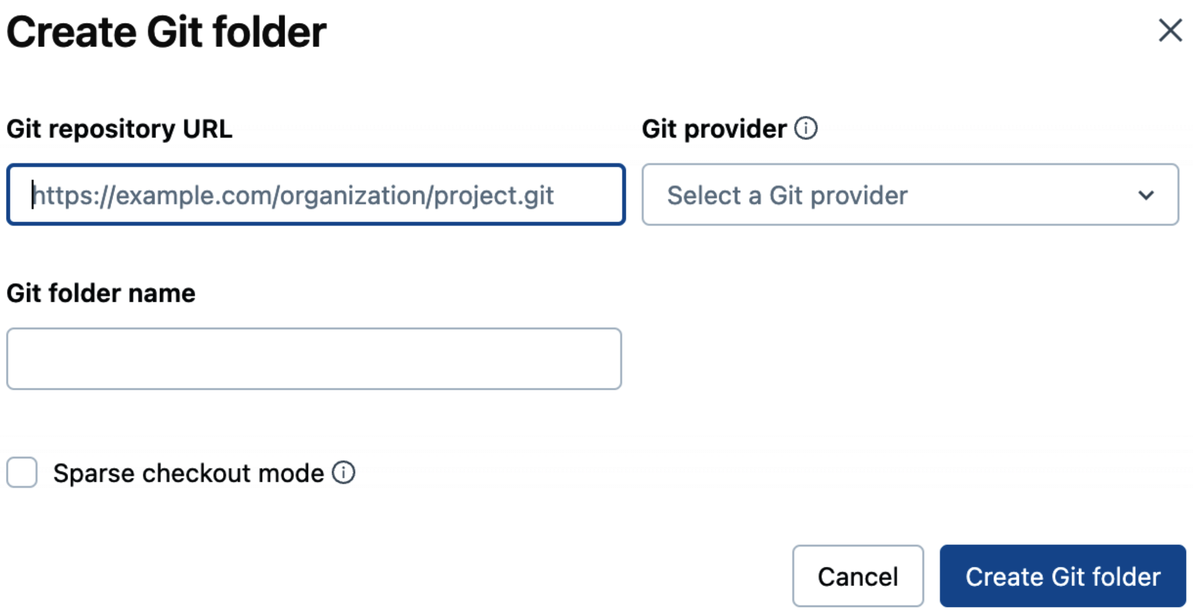


Figure 1-10. Git folder creation dialog

3. Paste the GitHub repository URL: In the Git folder creation dialog, paste the URL of the book's GitHub repository (*https://github.com/derar-alhussein/oreilly-databricks-dea*). The interface will automatically detect the Git provider (e.g., GitHub) and fill in the repository name based on the URL provided.
4. Create the Git folder: After confirming the details, click the Create Git folder button to clone the repository. The cloned repository will then appear as a folder in your workspace, and you can navigate through its contents just as you would with any other directory.

It's important to note that this feature is available only in the full version of Databricks. Users of the Databricks Community Edition should refer to the alternative method outlined next.

Option 2: DBC files

On the Databricks Community Edition, Git integration through Git folders is not supported. However, you can still import the book's materials by utilizing DBC files, which are archive files designed for directly importing source code into Databricks workspaces. To import the DBC file of our book's resources, follow these steps:

1. Download the DBC file from GitHub: Navigate to the book's GitHub repository and locate the *Exports* folder. From this folder, download the file named *book_materials.dbc* to your local machine.
2. Navigate to your workspace browser: In your Community Edition workspace, navigate to the Workspace tab to access your Home directory.
3. Use the Import option: At the top of your directory, click the menu icon (represented by three vertical dots) and select the Import option, as illustrated in Figure 1-11.

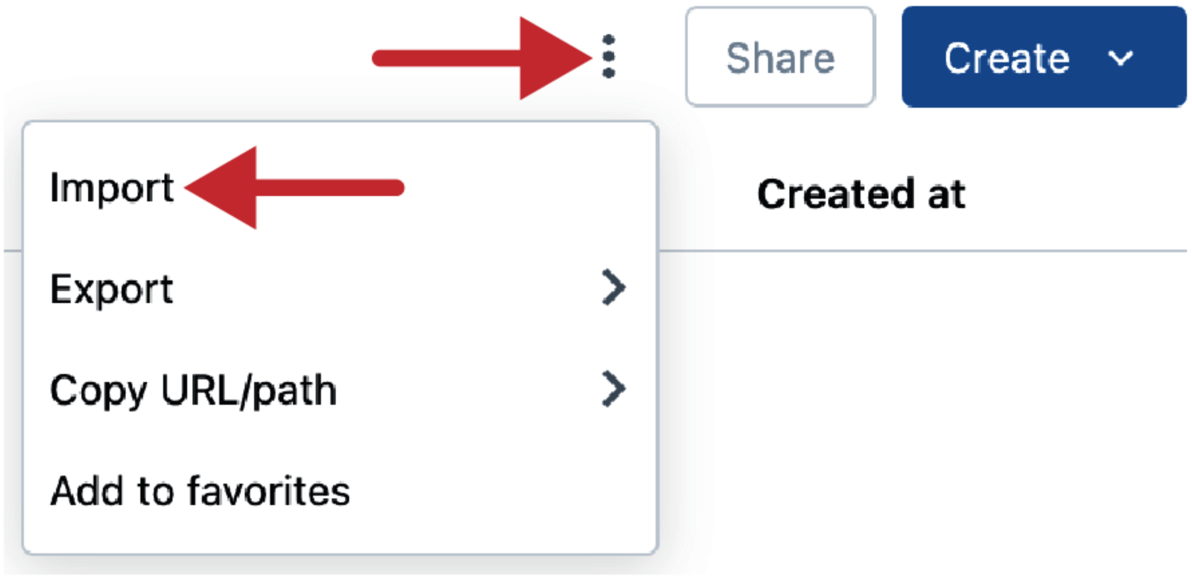
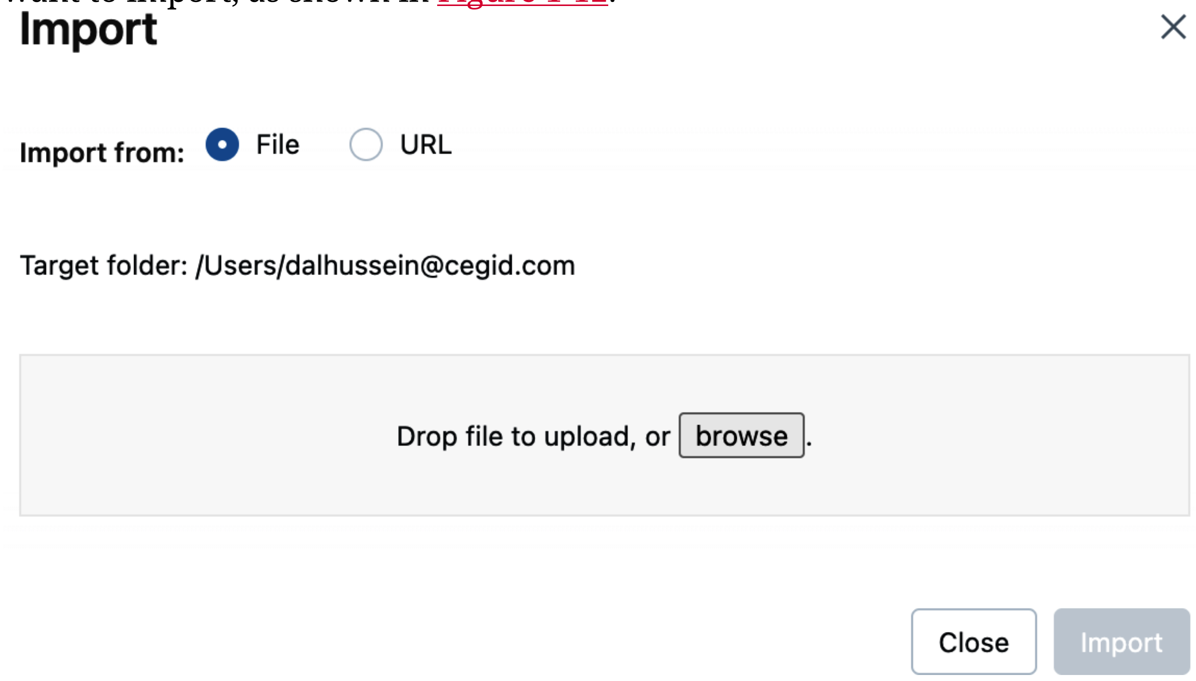


Figure 1-11. Importing files using the menu icon in the workspace browser

This action will open a dialog box where you can specify the file you want to import, as shown in Figure 1-12.



Target folder: /Users/dalhussein@cegid.com

Figure 1-12. File import dialog

4. Upload the DBC file: In the Import dialog, browse to select the DBC file you downloaded earlier and import it into your workspace.

By following these steps, you will have all the book’s resources in your workspace, allowing you to replicate the solutions within your own environment.

Creating Clusters

Clusters in Databricks form the backbone of data processing and analytics on the platform. A cluster is essentially a collection of computers, often referred to as *nodes*, instances, or virtual machines, working together as a single entity. In the context of Apache Spark, which powers Databricks, a cluster comprises a master node known as the *driver* and several worker nodes, as illustrated in Figure 1-13. The driver node is primarily responsible for orchestrating the activities of the worker nodes, which execute tasks in parallel, thereby enabling efficient processing of large-scale data.

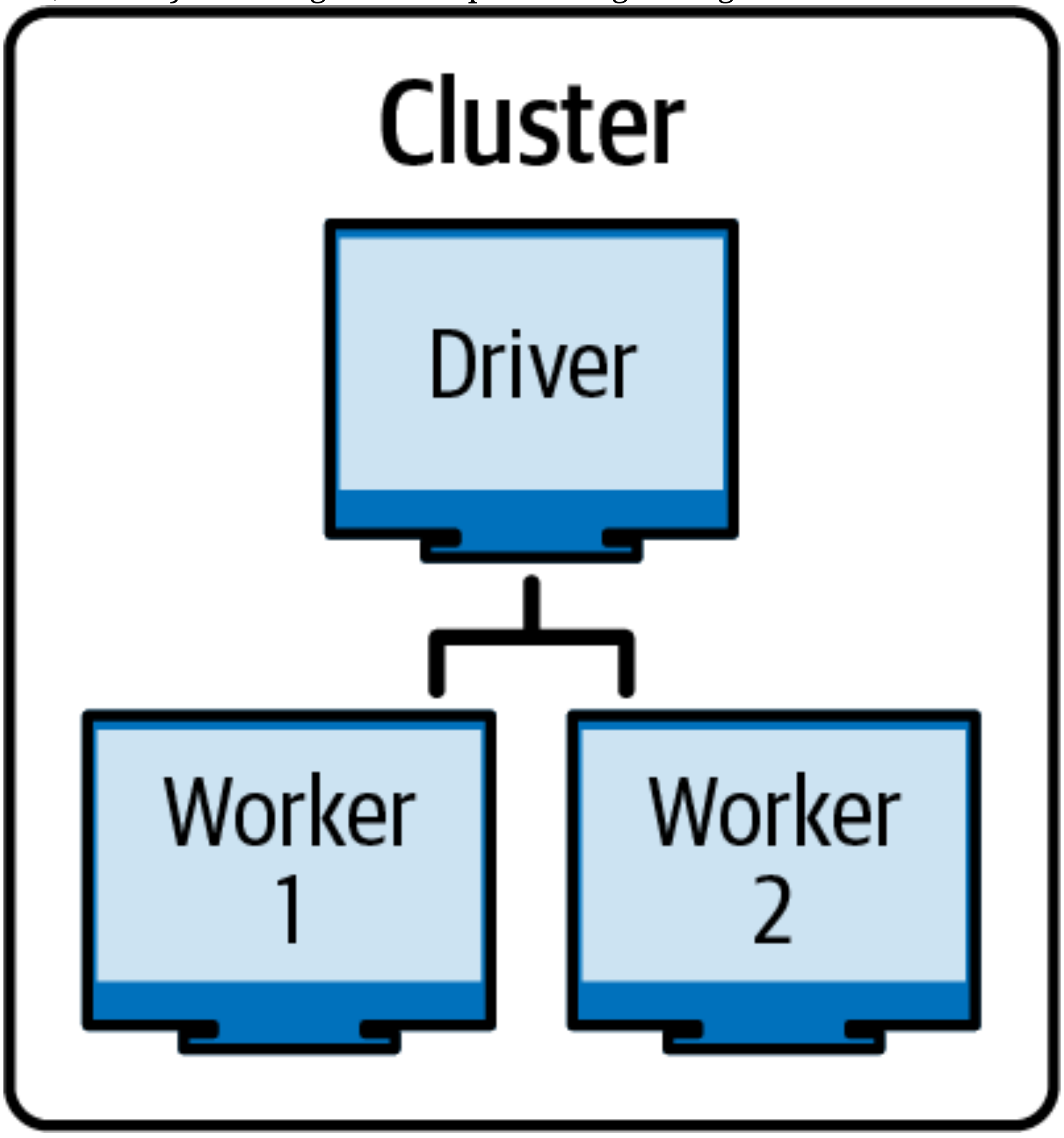


Figure 1-13. Apache Spark cluster architecture: driver node and worker nodes

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. Table 1-1 summarizes the differences between these two types of clusters.

	All-purpose cluster	Job cluster
Usage	Interactive development and data analysis	Automated job execution
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

Table 1-1. Comparison of all-purpose clusters and job clusters

Let’s dive deeper to gain a comprehensive understanding of these two types of clusters.

All-Purpose Clusters

All-purpose clusters are designed for interactive, exploratory tasks, making them ideal for development, and ad hoc analysis. They offer flexibility and control to users who need a dynamic environment to work with their data:

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.

In the next section, we will learn how to create and manage all-purpose clusters within the Databricks workspace. These clusters will be our primary tool for executing hands-on exercises throughout this book.

Job Clusters

Job clusters, on the other hand, are optimized for automated workloads. 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.

In Chapter 6, we explore job clusters in the context of Databricks Jobs and Delta Live Tables (DLT) pipelines.

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.

Understanding cluster pools

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.

Cost considerations

While cluster pools offer significant operational benefits, they come with important cost considerations. It’s essential to understand that even though Databricks itself does not charge for the idle instances in a pool, your cloud provider does. This is because these instances, although idle, are actively running on your cloud infrastructure, and as such, 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.

Creating All-Purpose Clusters

This guide walks you through the process of creating an all-purpose cluster, from initial navigation to final configuration.

1. Navigating to the Compute tab

To begin, access the Compute tab from the left sidebar in your Databricks workspace. This page is the central hub for managing all your Databricks clusters, as illustrated in Figure 1-14.

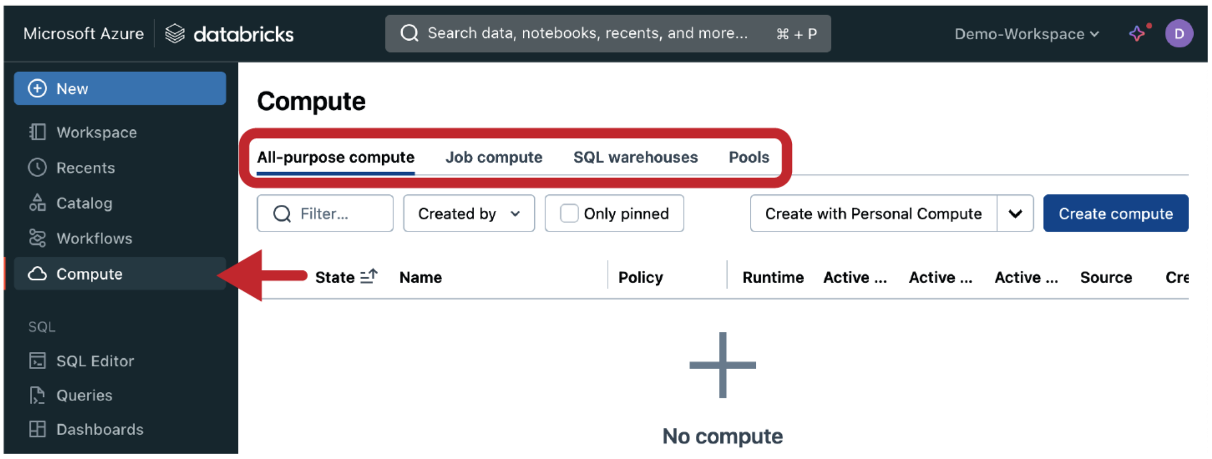
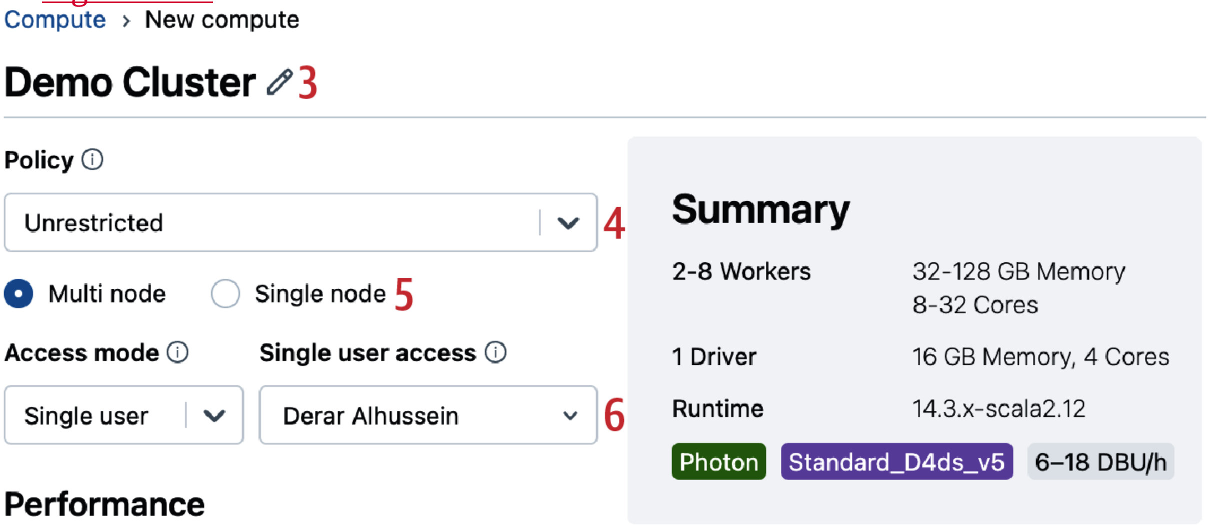


Figure 1-14. Compute page in the Databricks workspace

This page presents various tabs at the top, including “All-purpose compute,” “Job compute,” and Pools, each corresponding to the different types of compute resources discussed earlier. Additionally, there’s a tab for SQL warehouses, which are dedicated compute resources for executing SQL workloads within Databricks SQL. In Chapter 7, we delve deeply into Databricks SQL and explore the nuances of SQL warehouses. For the moment, let’s focus on setting up an all-purpose cluster to get started with running interactive workloads.

2. Initiating the cluster creation

Under the “All-purpose compute tab,” click the blue “Create compute” button. This action opens the configuration page for your new cluster, as illustrated in Figure 1-15.



Policy	Unrestricted
Multi node / Single node	[✓ Multi node] Single node
Access mode	Single user
Single user access	Derar Alhussein

2-8 Workers	32-128 GB Memory 8-32 Cores
1 Driver	16 GB Memory, 4 Cores
Runtime	14.3.x-scala2.12
Photon	Standard_D4ds_v5 · 6–18 DBU/h

Figure 1-15. Compute cluster configuration page

3. Naming your cluster

The first step in the configuration process is naming your cluster. The system provides a default name, but you can change it by clicking the name field at the top. For example, you might name your cluster “Demo Cluster” to reflect its purpose.

4. Setting the cluster policy

Next, you’ll encounter the Policy setting. By default, this is set to Unrestricted, allowing you full control over the cluster’s configuration. In environments with stricter governance or where specific configurations are mandated, other policies may be in place to limit certain settings.

5. Configuring the cluster: Single-node versus multi-node

Databricks allows you to choose between creating a single-node cluster or a multi-node cluster:

Single-node cluster
: This cluster operates with just a driver node, eliminating the need for additional worker nodes. In this configuration, the driver handles both driver and worker responsibilities, executing all Spark jobs on a single machine. This setup is more cost effective as it consumes fewer resources.

Multi-node cluster
: If you need to handle larger datasets or more complex processing tasks, you can opt for a multi-node cluster, which includes one driver node and multiple worker nodes. This setup allows parallel processing, making it suitable for heavier workloads.

For demonstration, this guide will focus on configuring a multi-node cluster to showcase the advanced configuration options available.

6. Configuring the access mode

Databricks clusters offer different access modes depending on how the cluster is intended to be used:

Shared access mode
: This allows multiple users to share the cluster simultaneously but restricts workloads to SQL and Python only. Shared clusters are useful in collaborative environments where several users need to access the same cluster.

Single user mode

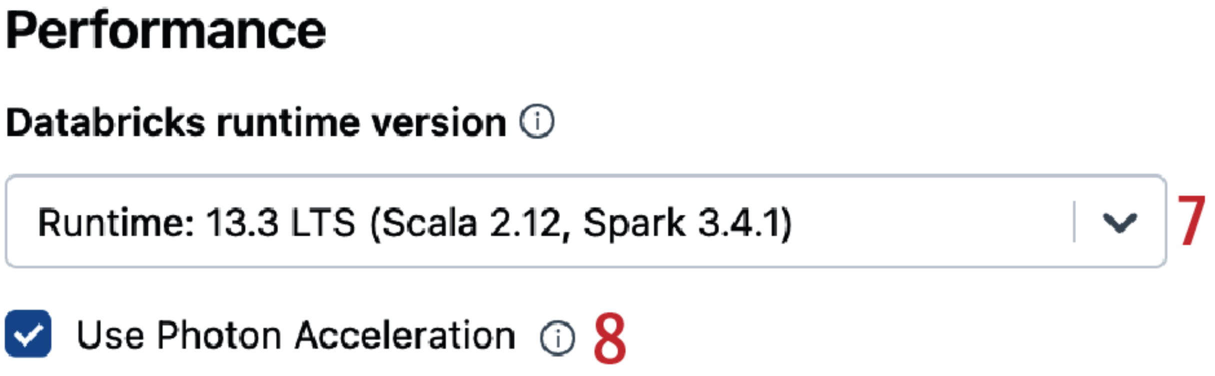
This mode is appropriate if you are the only one using the cluster. It ensures that the cluster resources are dedicated solely to your tasks, potentially improving performance and efficiency.

For this demonstration, select “Single user” mode as you are the only user of this cluster.

7. Performance: Selecting the Databricks Runtime version

The Databricks Runtime version is a critical choice, as it determines the software environment in which your clusters will operate. Databricks Runtime is a pre-configured virtual machine image that includes specific versions of Apache Spark, Scala, and various other libraries essential for data processing.

For this guide, choose Databricks Runtime 13.3 LTS (long-term support), as illustrated in Figure 1-16. This runtime version aligns with the version covered in the latest certification exam. Newer versions might offer additional features and optimizations that are not yet included in the exam. Sticking to the recommended version ensures you’re studying the relevant content for the test.


Performance

Databricks runtime version

Runtime: 13.3 LTS (Scala 2.12, Spark 3.4.1)

☑ Use Photon Acceleration

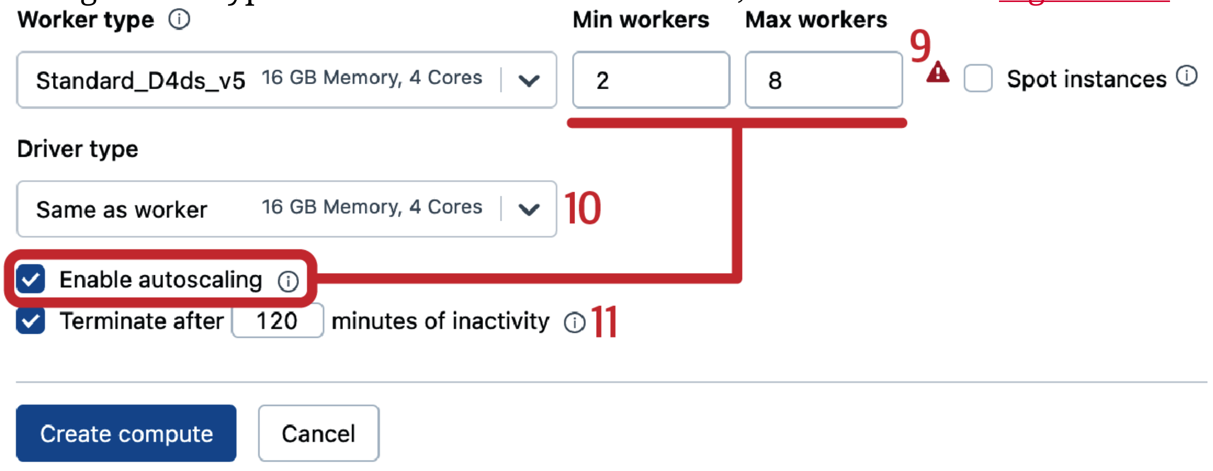
Figure 1-16. Compute cluster configuration page (continued)

8. Enabling Photon

Photon is an optional feature you can enable to further enhance your cluster’s performance. Photon is a high-speed query engine developed in C++, designed to accelerate the execution of SQL queries in Spark. Enabling Photon is particularly beneficial for workloads that involve heavy SQL processing or operations with many files, as it can significantly reduce query execution times and enhance overall performance. However, it’s essential to consider the additional costs associated with this feature.

9. Configuring worker nodes

Worker nodes are the backbone of a multi-node cluster, responsible for processing the distributed tasks assigned by the driver node. Here, you'll configure the type and number of worker nodes, as illustrated in Figure 1-17.


Worker type: Standard_D4ds_v5 16 GB Memory, 4 Cores

Min workers: 2

Max workers: 8

☐ Spot instances

Driver type: Same as worker 16 GB Memory, 4 Cores

☑ Enable autoscaling

☑ Terminate after 120 minutes of inactivity

Figure 1-17. Compute cluster configuration page (continued)

VM size selection
: Databricks allows you to choose from various virtual machine types and sizes provided by your cloud provider (e.g., Azure). These differ in terms of CPU cores, memory, and storage options, which should be selected based on the specific demands of your workloads. For simplicity, you may choose to keep the default VM size.

Number of workers
: Databricks offers an autoscaling feature, which dynamically adjusts the number of workers based on the cluster's workload. Enabled by default, the "Enable autoscaling" option allows you to specify a minimum and maximum range for the number of workers. Databricks will automatically increase or decrease the number of worker nodes within this range based on demand. Alternatively, you can disable autoscaling and set a fixed number of workers, such as 3, ensuring that the cluster always operates with the specified resources regardless of changes in workload.

10. Configuring the driver node

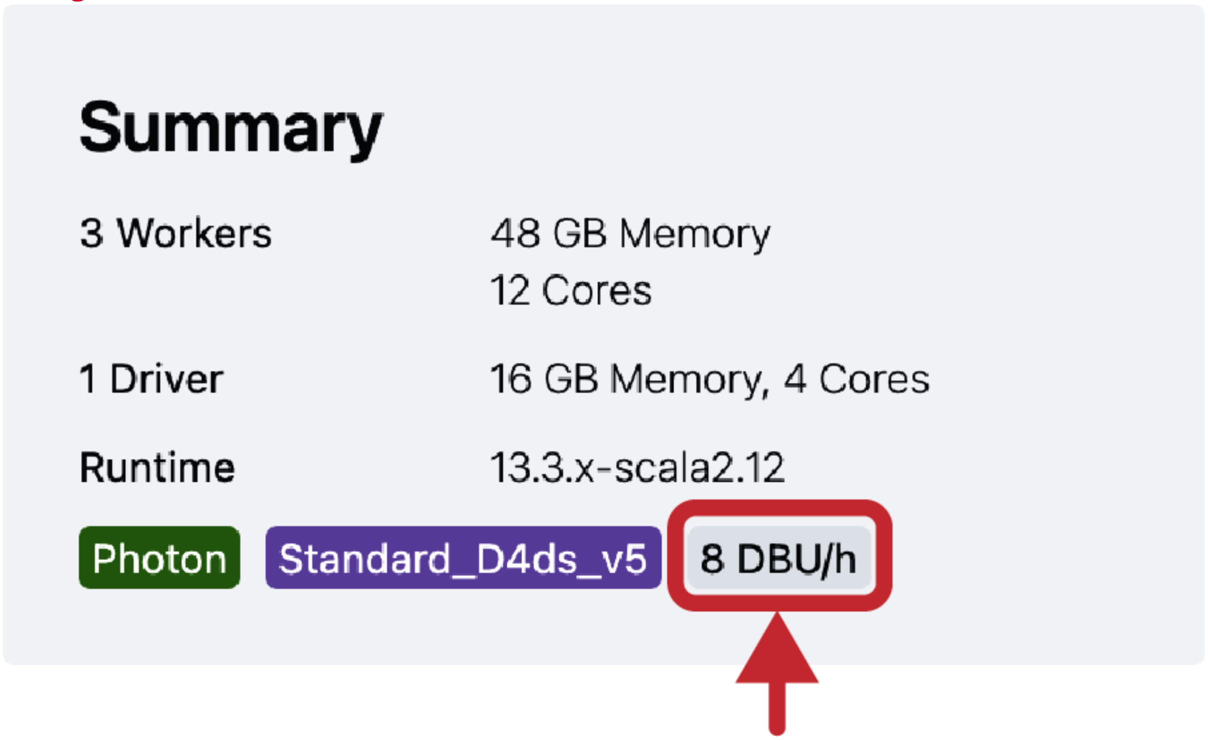
After configuring the worker nodes, you can set the configuration for the driver node, which coordinates all tasks across the cluster. You can either choose a different configuration for the driver or simply match it with the worker nodes, depending on your workload requirements.

11. Enabling auto-termination

To manage costs and optimize resource usage, Databricks provides an auto-termination feature, which is also enabled by default. By setting a specific duration of inactivity (e.g., 30 minutes), you can ensure that the cluster automatically shuts down if it remains idle for that period. This feature is particularly useful in preventing unnecessary charges for clusters that are no longer in use.

12. Reviewing the cluster configuration

As you configure the cluster, Databricks provides a summary on the right side of the screen, giving you a clear overview of your selections, as shown in Figure 1-18.



Summary

3 Workers	48 GB Memory 12 Cores
1 Driver	16 GB Memory, 4 Cores
Runtime	13.3.x-scala2.12

Photon Standard_D4ds_v5 8 DBU/h

Figure 1-18. Cluster configuration summary

This summary includes important details such as the total number of worker cores and RAM, the runtime version, and the number of Databricks units (DBUs) the cluster will consume. A DBU is a measure of processing capacity per hour, which helps estimate the costs associated with running the cluster. For example, a single-node cluster will generally consume fewer DBUs compared to a multi-node cluster, making it a more cost-effective choice for less demanding workloads. For precise DBU pricing specific to your cloud and region, consult the Databricks pricing page.

13. Creating the cluster

Once you have reviewed the configuration and ensured that it meets your needs, click the Create button. Databricks will then proceed to provision the required virtual machines, apply the configurations, and install Databricks Runtime and any additional specified libraries.

NOTE

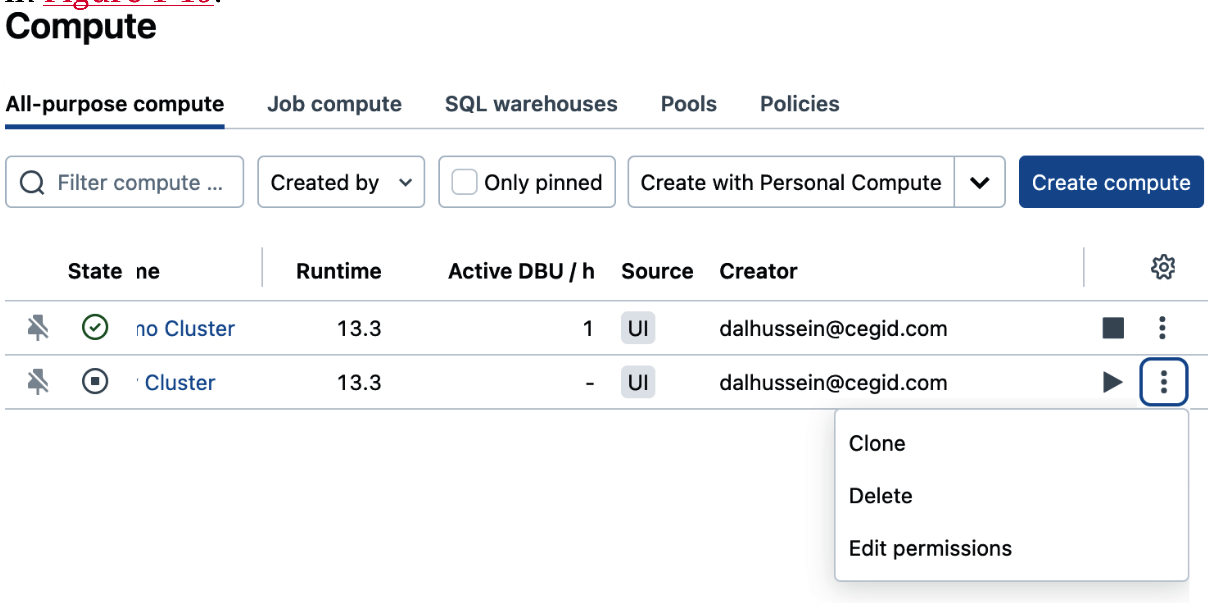
If you are using the free tier on Microsoft Azure cloud, there is a compute limit of four cores. To avoid a quota exceeded error, ensure that you use a single-node cluster with a maximum of four cores.

Managing Your Cluster

Once your cluster in Databricks is provisioned and running, indicated by a fully green circle next to its name, you have several options for managing and monitoring it.

Controlling your cluster

To access your cluster at any time, simply navigate to the Compute tab in the left sidebar of your Databricks workspace. This page lists all your clusters, displaying their current status, whether running or terminated, as illustrated in Figure 1-19.



State	ne	Runtime	Active DBU / h	Source	Creator
✓	no Cluster	13.3	1	UI	dalhussein@cegid.com
■	Cluster	13.3	-	UI	dalhussein@cegid.com

Figure 1-19. Compute page in the Databricks workspace

From this page, you can quickly start or terminate your cluster by clicking the play/stop button located on the right side of each cluster's entry. Next to this button is a menu icon represented by three vertical dots. Clicking this icon opens a drop-down menu with additional management features, including cloning the cluster, editing its permissions, and deleting it.

By clicking the cluster's name, you can access the configuration settings and make adjustments as needed. For example, you might want to change the instance type, adjust the number of workers, or enable additional features like Photon. However, be aware that changing the cluster configuration may require a restart of the cluster, which interrupts any running jobs.

Managing your cluster

Effective cluster management goes beyond just starting and stopping the cluster. Databricks provides tools to monitor the cluster's activity and troubleshoot any issues. These tools are accessible from the cluster configuration page, as illustrated in Figure 1-20.

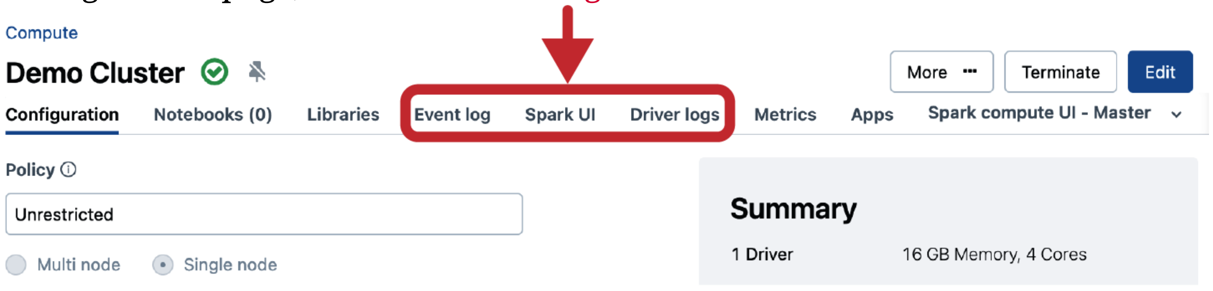


Figure 1-20. Compute cluster configuration page

Event log
: The "Event log" records all significant actions related to the cluster, such as when the cluster was created, terminated, edited, or encountered any errors. This detailed tracking enables effective monitoring and troubleshooting of cluster activities.

Spark UI
: The Spark UI provides a comprehensive interface for monitoring and debugging Apache Spark applications. It provides detailed insights into job execution, stages, and tasks that enable you to easily track performance and identify bottlenecks.

Driver logs
: "Driver logs" contains logs generated by the driver node within the cluster. This log captures output from the notebooks and libraries running on the cluster, making it an essential tool for diagnosing and resolving issues during development.

With our cluster up and running, we're now ready to execute code within Databricks notebooks. This will be the focus of the following section.

Working with Notebooks

Databricks notebooks are interactive development environments that enable you to write, debug, and execute code in a collaborative setting. These notebooks offer advanced capabilities that extend beyond those of traditional environments like Jupyter Notebooks. Databricks notebooks support multiple programming languages, including Python, SQL, Scala, and R. In addition, they integrate seamlessly with Spark clusters, allowing users to leverage distributed computing resources directly from the notebook interface.

Creating a New Notebook

To begin working with notebooks in Databricks, navigate to the Workspace tab in the left sidebar of your Databricks workspace. To create a new notebook, click the blue Create button and choose Notebook from the drop-down menu, as illustrated in Figure 1-21.

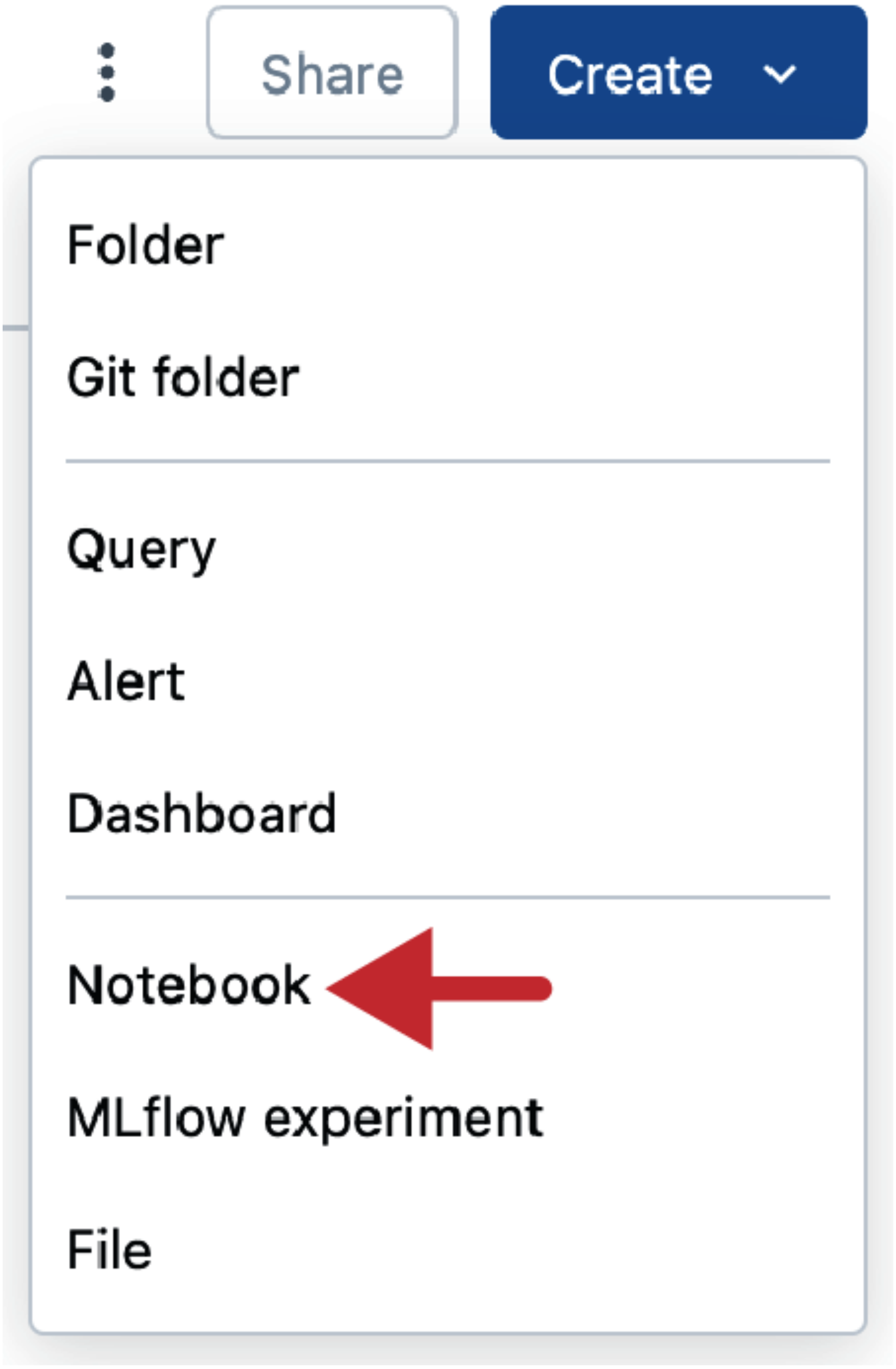
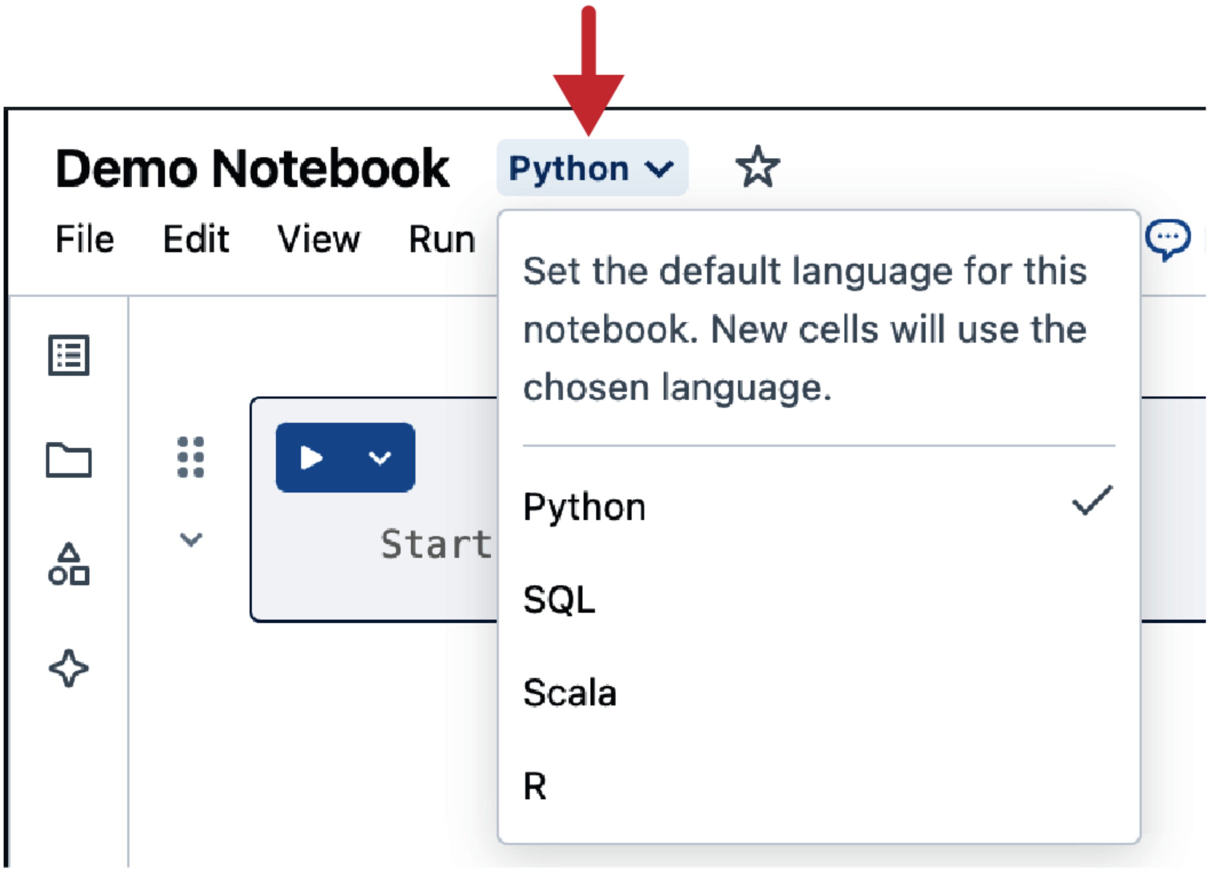

Share
Create
Folder
Git folder
Query
Alert
Dashboard
Notebook
MLflow experiment
File

Figure 1-21. Adding new notebooks using the Create button in the workspace browser

This action will create and open a new notebook, initially named “Untitled Notebook.” The notebook is immediately ready for use, but it’s good practice to rename it to something more descriptive. To rename your notebook, simply click the title at the top of the notebook interface and enter a new name, such as `Demo Notebook`.

Setting the Notebook Language

Databricks notebooks default to Python, but they support multiple languages, including SQL, Scala, and R. If you need to work in a language other than Python, you can easily change the notebook’s default language. To do this, click the language indicator at the top of the notebook, where it says “Python,” and select the desired language from the drop-down menu, as shown in Figure 1-22.



Set the default language for this notebook. New cells will use the chosen language.

Python ✓

SQL

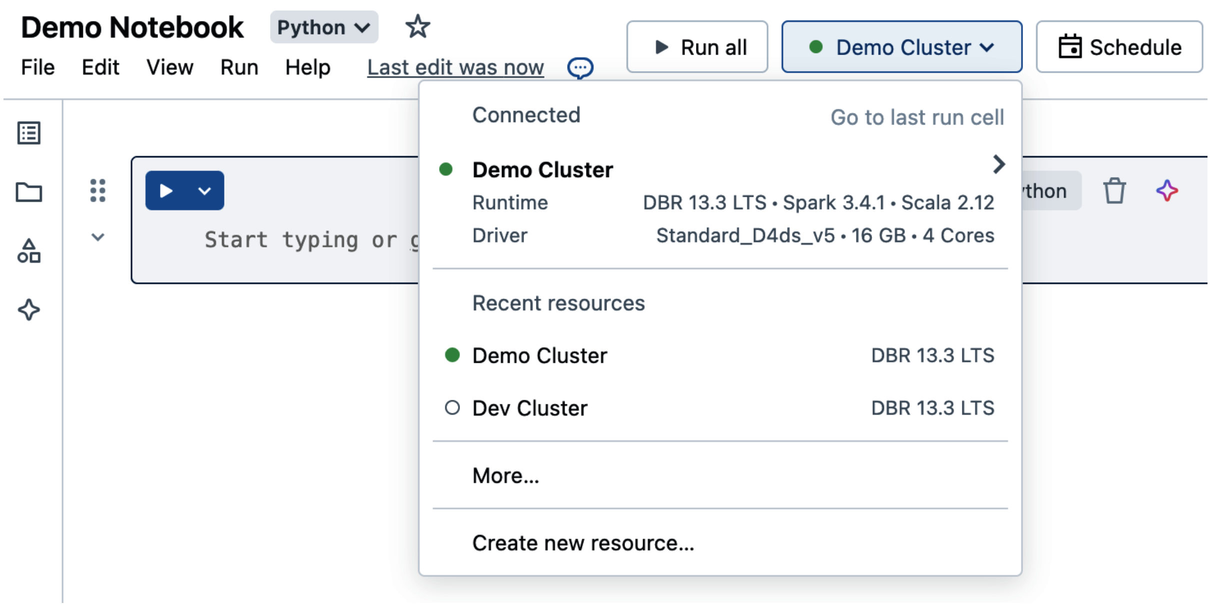
Scala

R

Figure 1-22. Changing the default language in Databricks notebooks

Executing Code

Before executing any code, it’s necessary to connect your notebook to an active cluster. Click the Connect button in the top-right corner of the notebook interface, as illustrated in Figure 1-23, and select the desired cluster, such as Demo Cluster created earlier.



Runtime	DBR 13.3 LTS • Spark 3.4.1 • Scala 2.12
Driver	Standard_D4ds_v5 • 16 GB • 4 Cores

Recent resources

● Demo Cluster	DBR 13.3 LTS
○ Dev Cluster	DBR 13.3 LTS

Figure 1-23. Attaching a cluster to a Databricks notebook

If the cluster is currently terminated, selecting it will automatically start it. Starting a terminated cluster can take a few minutes, depending on factors like the cluster's size and configuration.

NOTE

Databricks offers serverless compute for notebooks, which allows you to run code without the need to configure and deploy infrastructure. To use this feature, your workspace must be enabled for Unity Catalog, and serverless compute must be activated in your Databricks account.

Once the cluster is running, indicated by a fully green circle next to its name, your notebook is ready to execute code.

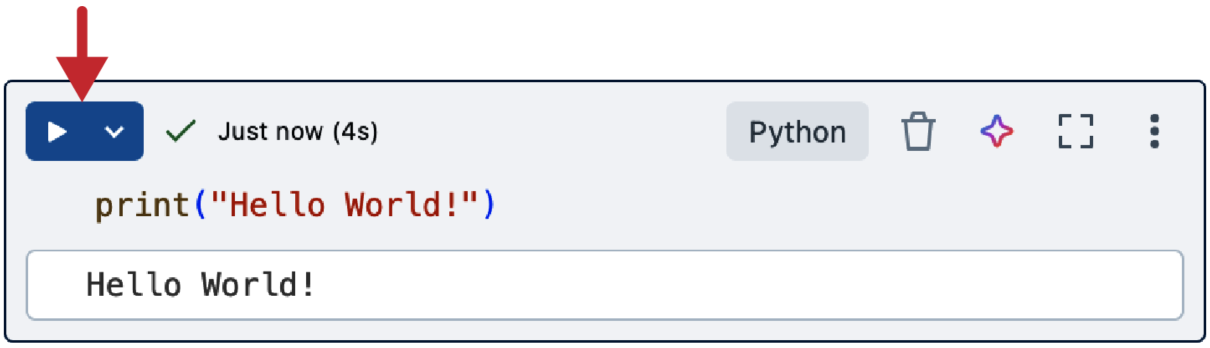
Running code cells

Databricks notebooks use a cell-based structure, where each cell can contain a block of code. This structure allows for interactive development, where you can run each cell independently, see immediate results, and make adjustments as needed.

Let's start by printing a simple "Hello World" message in our notebook. To do this, enter the following Python command into the first cell:

```
print("Hello World!")
```

To run the cell, click the play button on the left side of the cell, as displayed in Figure 1-24. Alternatively, you can use the Shift+Enter keyboard shortcut to run the current cell and move to the next one. This tends to be more efficient, especially when running lots of cells in succession.



```
print("Hello World!")
```

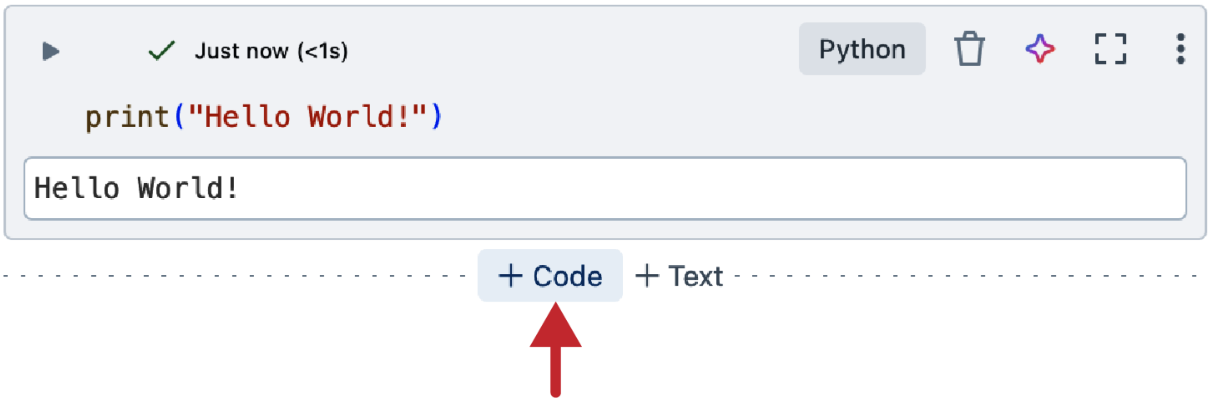
```
Hello World!
```

Figure 1-24. Running code cells in Databricks notebooks

The output of the cell—in this case, “Hello World!”—will appear directly below the cell. This immediate feedback is one of the key advantages of working with notebooks, allowing you to experiment and iterate quickly. Other types of outputs can have richer displays, and you’ll also see error messages, tool tips, or other warnings here as you work.

Managing cells

Notebooks provide a flexible environment for organizing your code. You can add, move, and remove cells to structure your code logically. To add a new cell, hover your mouse just below an existing cell, and you’ll see a + Code button appear, as illustrated in Figure 1-25. Click this button to insert a new cell.



```
print("Hello World!")
```

```
Hello World!
```

Figure 1-25. Adding a new cell in Databricks notebooks

This approach allows you to break your code into manageable sections, making it easier to develop, debug, and maintain.

Magic Commands

Magic commands in Databricks notebooks are special cell instructions that provide additional functionality in the notebook environment. These commands, which are prefixed with a `%`, allow you to execute tasks that go beyond standard code execution. Let’s explore these commands in detail, highlighting their benefits and how to use them effectively.

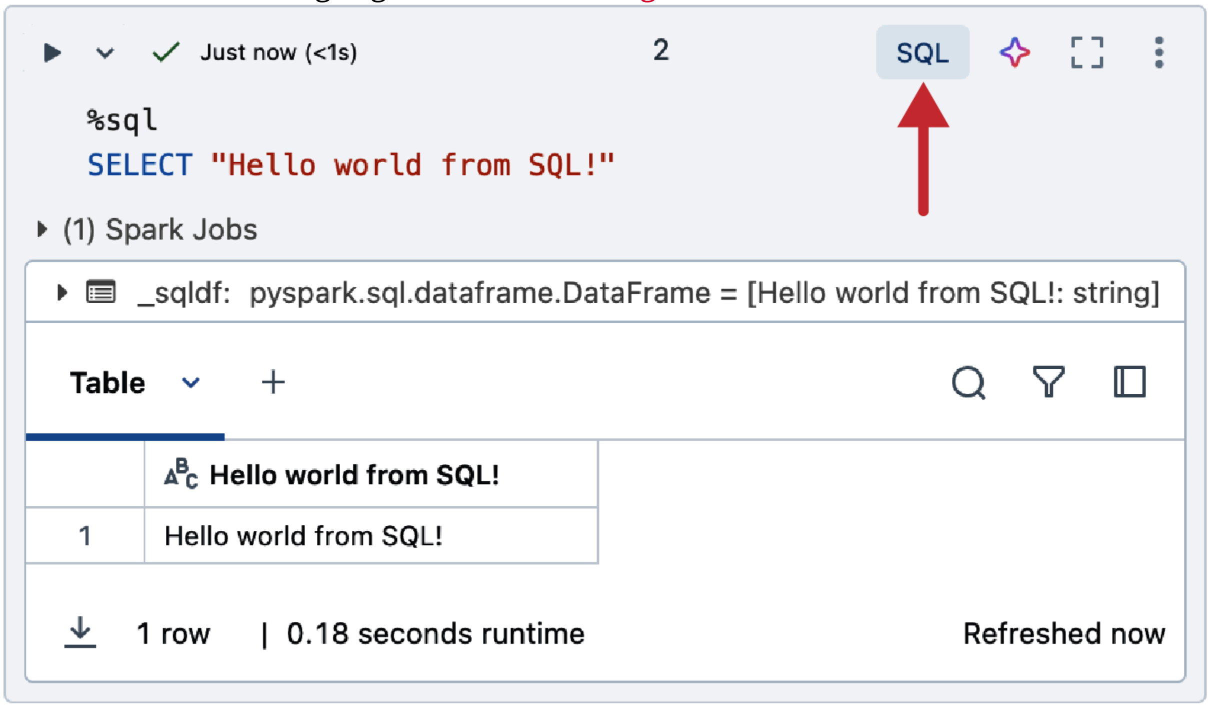
Language magic command

By default, a notebook is set to one primary language, but you may often need to use a different language within the same notebook for specific tasks. Language magic commands allow you to execute code cells in a language other than the notebook's default without changing the entire notebook's settings.

To switch the language for a specific cell, you just need to add the language magic command at the beginning of the cell. For example, if your notebook's default language is Python, but you need to run a SQL query, you would use the `%sql` magic command. This command instructs the notebook to interpret and execute the cell as SQL code:

```
%sql
SELECT "Hello world from SQL!"
```

When you enter a SQL query in a cell, Databricks automatically prepends the `%sql` magic command if the cell's content is detected as SQL. The cell's language indicator, located on the right side of the cell, will also change to reflect the active language, as shown in Figure 1-26.



```
%sql
SELECT "Hello world from SQL!"
```

```
_sqldf:  pyspark.sql.dataframe.DataFrame = [Hello world from SQL!: string]
```

	Hello world from SQL!
1	Hello world from SQL!

Figure 1-26. Language magic command in Databricks notebooks

If you want to manually change the language of a cell or convert it to a text Markdown cell, you can also do so by clicking this language indicator. This action will bring up a drop-down menu that allows you to select the desired option.

Markdown magic command

Beyond code execution, Databricks notebooks support rich-text formatting through the use of Markdown, which is enabled by the `%md` magic command. Markdown is an annotation language that allows you to format text and insert elements such as images or links directly within the notebook. This feature is particularly useful for documenting your analysis, adding notes, or structuring your notebook into sections.

To add formatted text, simply start a cell with the `%md` magic command, followed by your Markdown syntax. For instance, to create headers of different levels, you might use the following commands:

```
%md
# Title 1
## Title 2
### Title 3
```

When you press Esc, the text will be rendered as headers with varying levels of emphasis, as illustrated in Figure 1-27.

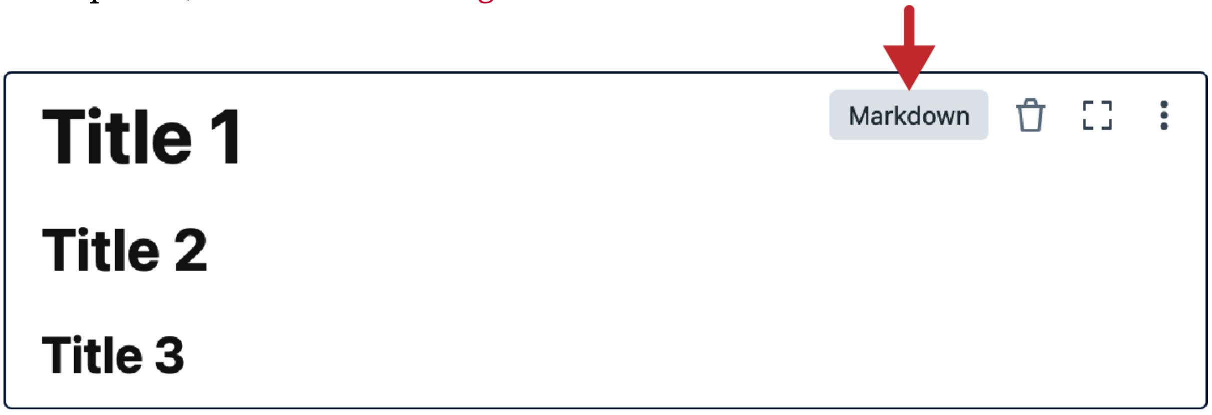


Figure 1-27. Markdown magic command in Databricks notebooks

Double-clicking a Markdown cell reopens its editor, where you'll find a toolbar with a range of formatting options such as bold, italic, and list creation. It also offers the ability to add images and hyperlinks, which enriches your notes or documentation.

Enhancing notebook navigation with Markdown

One of the significant advantages of using Markdown headers in Databricks notebooks is that they automatically generate entries in the notebook's table of contents. The table of contents is a navigational aid that allows you to quickly jump between sections in your notebook.

To access the table of contents, click its icon in the left-hand panel of the notebook editor, as shown in Figure 1-28.

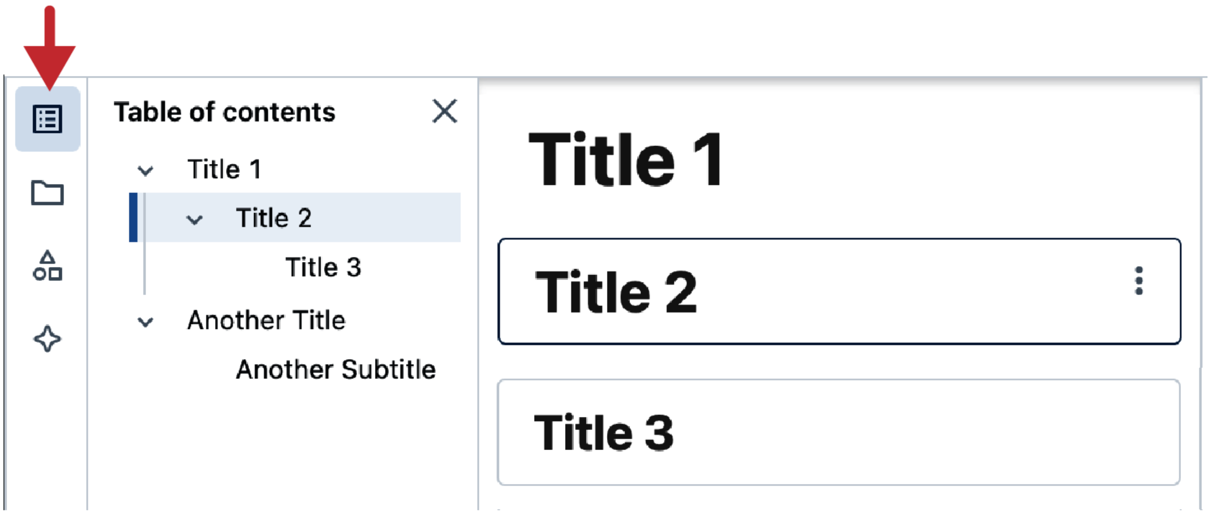


Figure 1-28. Table of contents in Databricks notebooks

As you add more Markdown headers, these will populate the table of contents, providing an organized overview of your notebook's structure.

Run magic command

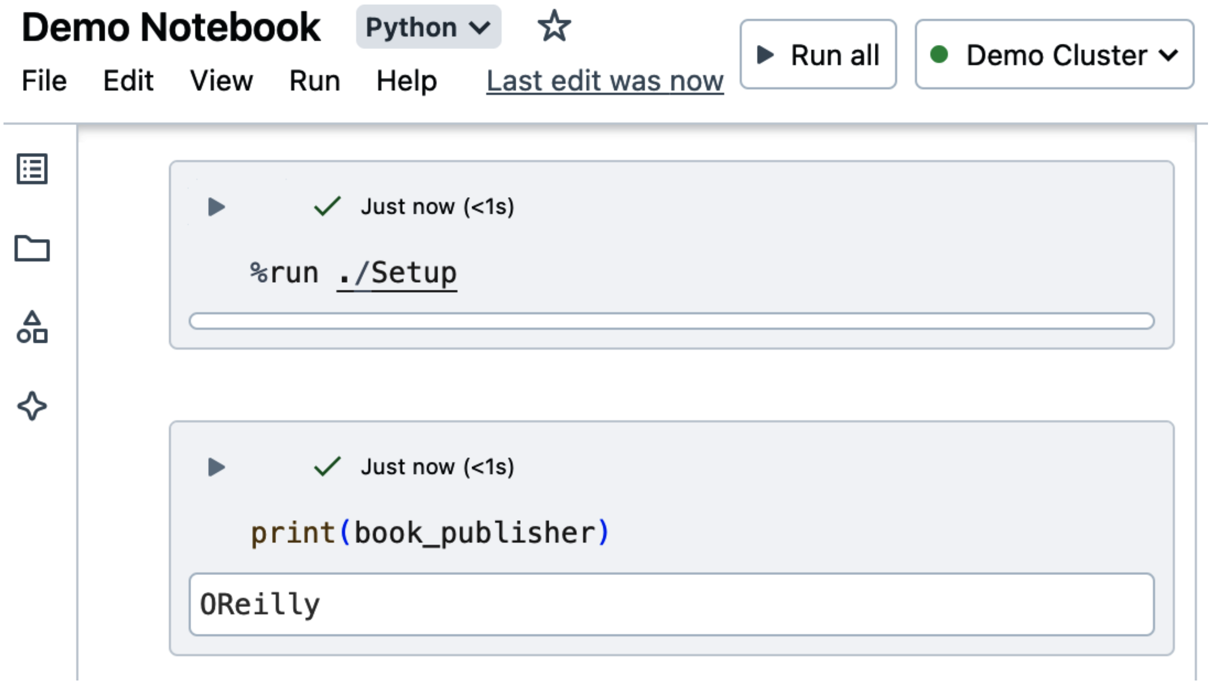
The `%run` magic command in Databricks notebooks is a powerful tool that allows you to execute another notebook within the current notebook. This feature is particularly useful for supporting code modularity and reusability. The `%run` command is designed to import and execute all the content of a specified notebook into the current notebook. This means that any variables, functions, or classes defined in the referenced notebook become accessible in the notebook that invoked the `%run` command. This is highly beneficial for scenarios where you need to share common configurations and functions across multiple notebooks.

To illustrate the use of the `%run` magic command, let's walk through a practical example where you have two notebooks—our primary notebook named "Demo Notebook" and a secondary notebook named "Setup":

1. Creating the Setup notebook: First, create a new notebook called "Setup" in your Home directory. In this notebook, you define a simple variable, `book_publisher`, and assign it the value `OReilly`:
   ```
   book_publisher = "OReilly"
   ```
2. Using the `%run` command: Now, switch to your demo notebook, where you want to access the variables of the Setup notebook. In a new cell, use the `%run` magic command followed by the path to the Setup notebook. Since both notebooks are in the same directory, you can use the dot symbol (.) to refer to the current directory, or alternatively, you could use the full workspace path to specify the exact location of the Setup notebook:
   ```
   %run ./Setup
   ```

3. Accessing the imported variable: After running the previous command, the entire contents of the Setup notebook are executed, bringing any defined variables and functions into the scope of the demo notebook. To verify this, you can print the `book_publisher` variable in a new cell in the demo notebook.

Figure 1-29 displays the output of the print command, confirming that the `book_publisher` variable was successfully imported from the Setup notebook into our demo notebook.



```
%run ./Setup
```

```
print(book_publisher)
```

```
OReilly
```

Figure 1-29. Run magic command in Databricks notebooks

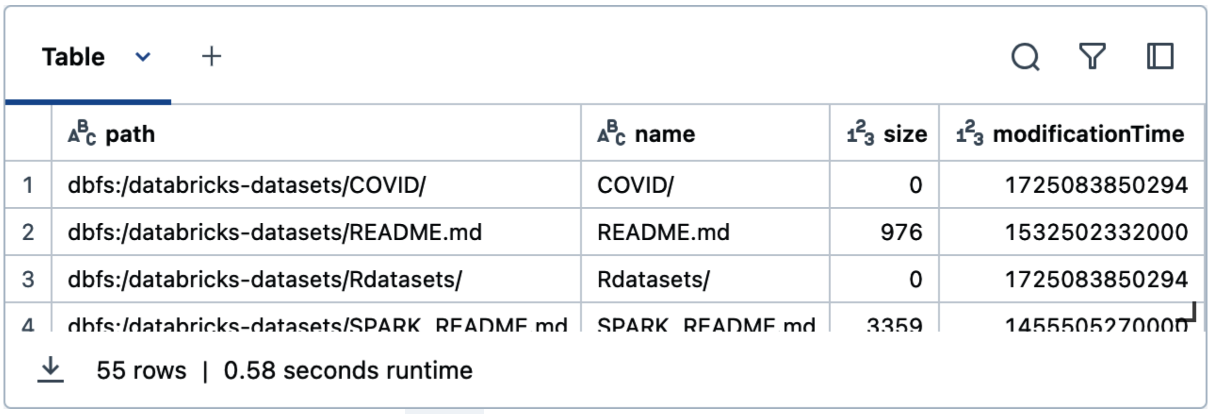
The `%run` magic command is an essential feature for anyone working with Databricks notebooks, offering a seamless way to create modular, maintainable, and reusable code.

FS magic command

When working within the Databricks environment, managing files and interacting with the file system is a common task. The `%fs` magic command provides a simple way to execute file system operations directly within your notebook cells. This command allows you to perform various tasks, such as copying, moving, and deleting files and directories within your cloud storage. One of the most common uses of the `%fs` magic command is listing the contents of a directory. For instance, if you want to explore the sample datasets directory provided by Databricks, you can use the following command:

```
%fs ls '/databricks-datasets'
```

Running this command will list all files and folders within the */databricks-datasets* directory, displaying 55 items by default, as illustrated in Figure 1-30.



Table

	path	name	size	modificationTime
1	dbfs:/databricks-datasets/COVID/	COVID/	0	1725083850294
2	dbfs:/databricks-datasets/README.md	README.md	976	1532502332000
3	dbfs:/databricks-datasets/Rdatasets/	Rdatasets/	0	1725083850294
4	dbfs:/databricks-datasets/SPARK_README.md	SPARK_README.md	3359	1455505270000

55 rows | 0.58 seconds runtime

Figure 1-30. Output of the `%fs ls` magic command on the /databricks-datasets directory

While the `%fs` magic command is convenient, Databricks provides a more flexible and powerful tool called `dbutils`. This tool is particularly useful for integrating file system operations directly into your Python code.

Databricks Utilities

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

To explore all available commands and their usage within `dbutils`, you can use the help function:

```
dbutils.help()
```

If you're interested in a specific utility within `dbutils`, you can request detailed help for that particular module. For example, if you want to learn more about the file system commands provided by `dbutils.fs`, you can use this:

```
dbutils.fs.help()
```

This command will provide information about the file system operations available, including how to perform common tasks such as listing directories.

To achieve similar functionality as the `%fs` command using `dbutils`, you can list the contents of the same directory with the following code:

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

This command not only lists the files but also stores the output in a variable (files), which can be further manipulated within your code.

Displaying the output

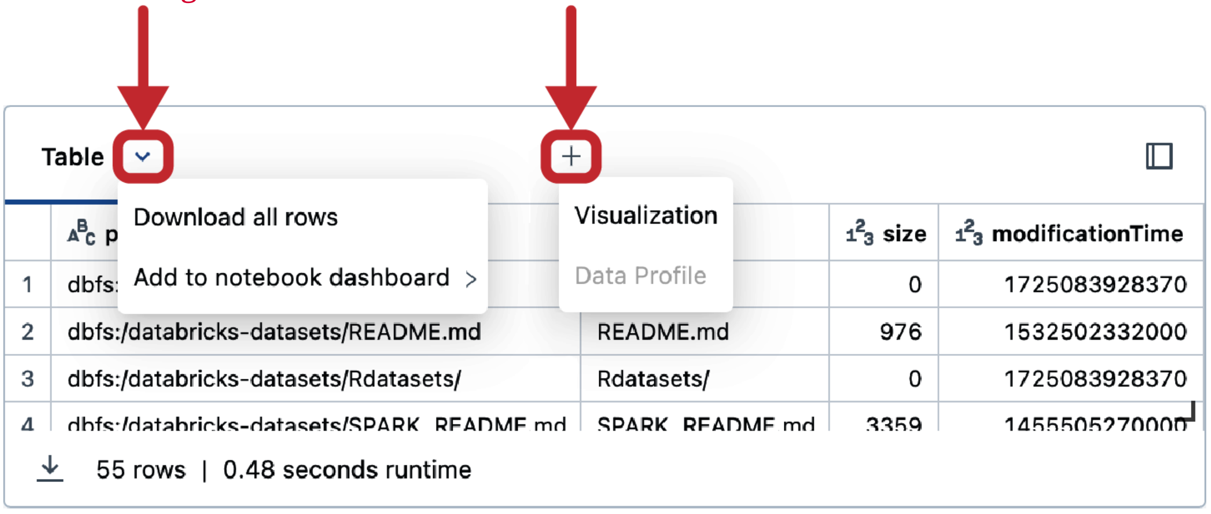
Directly printing the "files" variable might result in an output that's difficult to read:

```
print(files)
```

To present this information in a more user-friendly way, Databricks provides the display function, which formats the output in a tabular layout:

```
display(files)
```

Using this function, the results are neatly organized into columns, such as filename, size, and type, making it easier to understand and work with the data. Additionally, the display function offers advanced features, like downloading the data as a CSV file or visualizing the results in a graph, as shown in Figure 1-31.



	p		size	modificationTime
1	dbfs:		0	1725083928370
2	dbfs:/databricks-datasets/README.md	README.md	976	1532502332000
3	dbfs:/databricks-datasets/Rdatasets/	Rdatasets/	0	1725083928370
4	dbfs:/databricks-datasets/SPARK_README.md	SPARK_README.md	3359	1455505270000

55 rows | 0.48 seconds runtime

Figure 1-31. Advanced features of the preview display, including data download and visualization options

It’s important to be aware that while the display function is useful, it has limitations when previewing large datasets, as it shows only a subset of records.

Comparison: %fs magic command versus dbutils

Choosing between the `%fs` magic command and `dbutils` depends on the complexity and requirements of your task. 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.

Download Notebooks

You may want to download a notebook to your local system for various reasons, such as sharing with others or simply keeping a local copy. Databricks offers a straightforward way to export your notebooks by following these steps:

1. Navigate to the File menu: In the upper menu of your notebook editor, click the File menu, as illustrated in Figure 1-32.

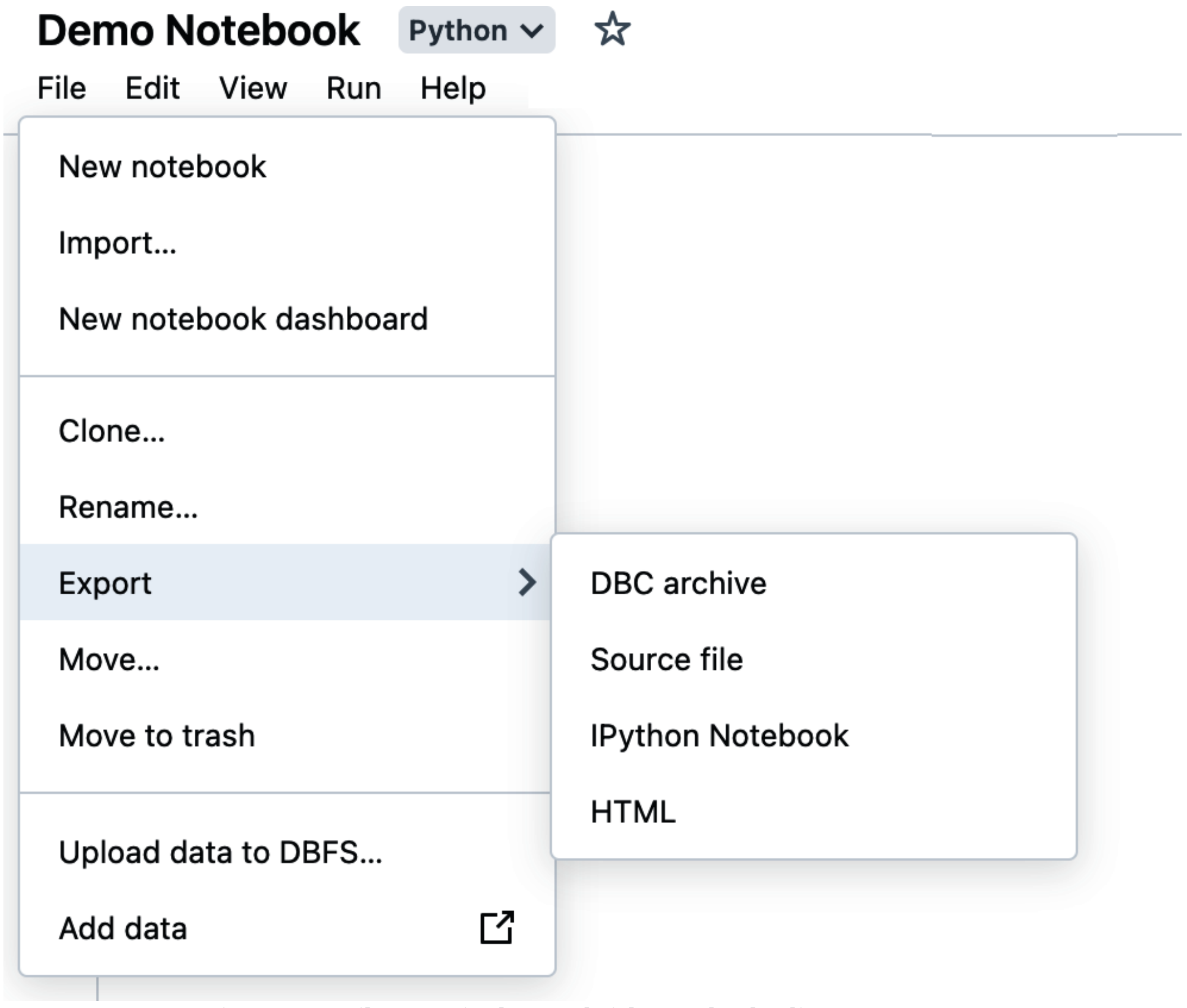


Figure 1-32. File menu in the Databricks notebook editor

2. Select Export: From the drop-down menu, choose the Export option. This will present you with several formats for downloading your notebook.
3. Choose file format: Click “Source file” to download the notebook as a plain Python script (*.py* file). For an HTML output, select IPython Notebook, which is useful for sharing results without needing to render them within a workspace.

The downloaded file can be edited locally or easily imported into a different Databricks workspace for further use. To import a file into a specific folder on Databricks, simply click the menu icon in the target folder and select the Import option, as shown in Figure 1-33.

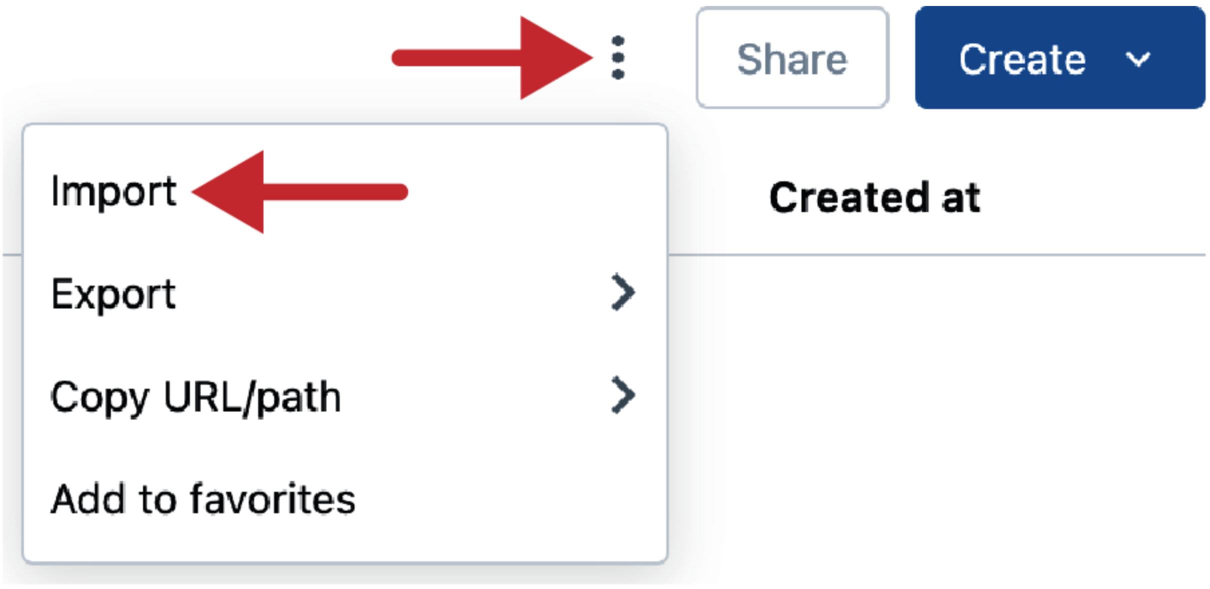


Figure 1-33. Importing files using the menu icon in the workspace browser

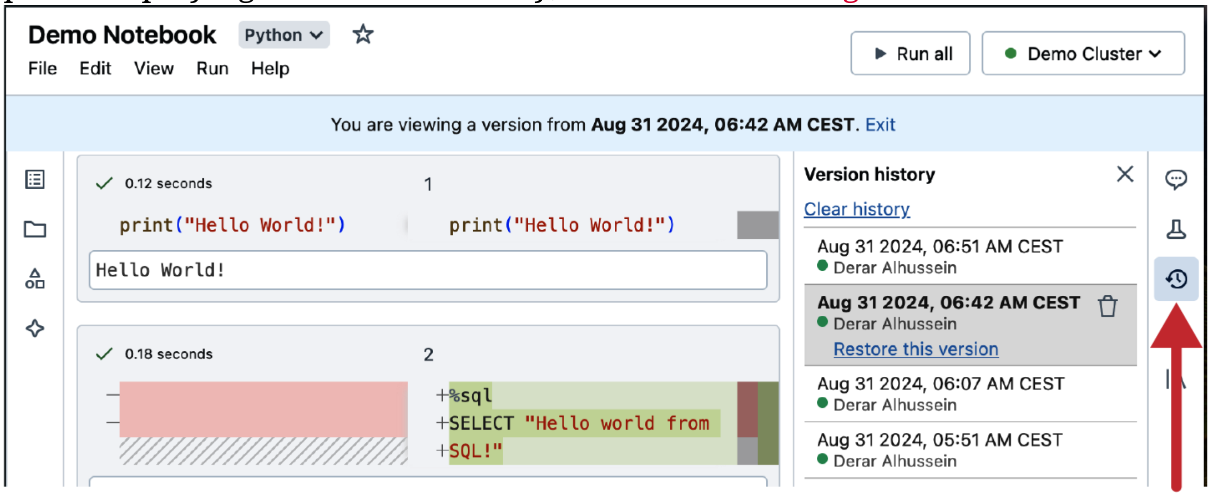
By following these steps, you can ensure that you have the flexibility to work with your notebooks across different environments.

Notebook Versioning

As you develop your code in Databricks, you will likely make numerous changes and refinements, which can be challenging to track. To address this issue, Databricks provides a built-in versioning system within its notebooks, allowing you to easily manage and revert to previous versions of your code.

Accessing version history

To access the version history of a notebook, look for the “Version history” icon located in the right sidebar of your notebook editor. Clicking this icon opens a panel displaying the version history, as illustrated in Figure 1-34.



You are viewing a version from **Aug 31 2024, 06:42 AM CEST**. Exit

```
0.12 seconds                1
print("Hello World!")       print("Hello World!")
Hello World!

0.18 seconds                2
                            +%sql
                            +SELECT "Hello world from
                            +SQL!"
```

Aug 31 2024, 06:51 AM CEST
● Derar Alhussein
Aug 31 2024, 06:42 AM CEST
● Derar Alhussein
Aug 31 2024, 06:07 AM CEST
● Derar Alhussein
Aug 31 2024, 05:51 AM CEST
● Derar Alhussein

Figure 1-34. Accessing notebook version history

This “Version history” panel shows a chronological list of all changes made to the notebook. Each entry in the list corresponds to an auto-saved version of the notebook, capturing the state of the notebook at that point in time.

Restoring a previous version

If you need to revert to a previous state of the notebook, simply select the desired version from the list and click “Restore this version.” This action reverts the current state of the notebook to the selected version, undoing any changes made since that version.

While this versioning feature is helpful for tracking changes, it has limitations, especially in complex or collaborative projects. It lacks advanced capabilities such as merging changes or creating branches. Additionally, users can easily delete this history, which may compromise its reliability. For a more robust solution, Databricks provides integration with Git providers, offering enhanced version control capabilities.

Versioning with Git

Databricks offers Git integration, allowing users to manage their data projects using familiar Git workflows, including branching, merging, committing, and pushing changes to remote repositories. This feature is particularly beneficial for users who need to manage complex projects, collaborate with team members, or maintain a history of changes in a more controlled and secure manner than what the basic notebook versioning can offer.

This seamless integration is facilitated through Git folders, formerly known as Databricks Repos, which enable source control directly into your Databricks workspace. With Git folders, you can synchronize your code with remote Git repositories and perform common Git operations.

Setting Up Git Integration

When importing source code from a public repository like our book’s GitHub repo, Git folders work seamlessly without additional setup. However, for private repositories or when performing Git operations like committing and pushing changes, you must link your Databricks workspace with your Git service provider. This setup ensures that you can perform all necessary Git operations from within Databricks.

Prerequisites

Before setting up Git integration, ensure the following:

Access to a full version of Databricks
The Git integration feature is not available in the Databricks Community Edition, so you'll need access to a full version of Databricks on a cloud platform like AWS, Azure, or Google Cloud.

Git service account
You should have an account with a supported Git service provider, such as GitHub or Azure DevOps.

Configuring Git integration

To configure Git integration in your Databricks workspace, follow these steps:

1. Access your profile settings: In the Databricks workspace, click your username profile icon located in the upper-right corner. From the drop-down menu, select Settings.
2. Link your Git provider: In the Settings page, navigate to the "Linked accounts" tab from the left side panel, as displayed in Figure 1-35. Here, you will find options to link your Databricks account with various Git service providers, including GitHub, Azure DevOps, and Bitbucket. Select your desired provider from the drop-down menu.
3. Authenticate your Git provider: If you are linking with GitHub, you can use a more secure method through the Databricks GitHub App, instead of using a personal access token (PAT). Select the "Link Git account" option, and click the Link button to start the process. You will be redirected to GitHub to authorize the Databricks app to access your GitHub account. Follow the on-screen instructions to complete the process.

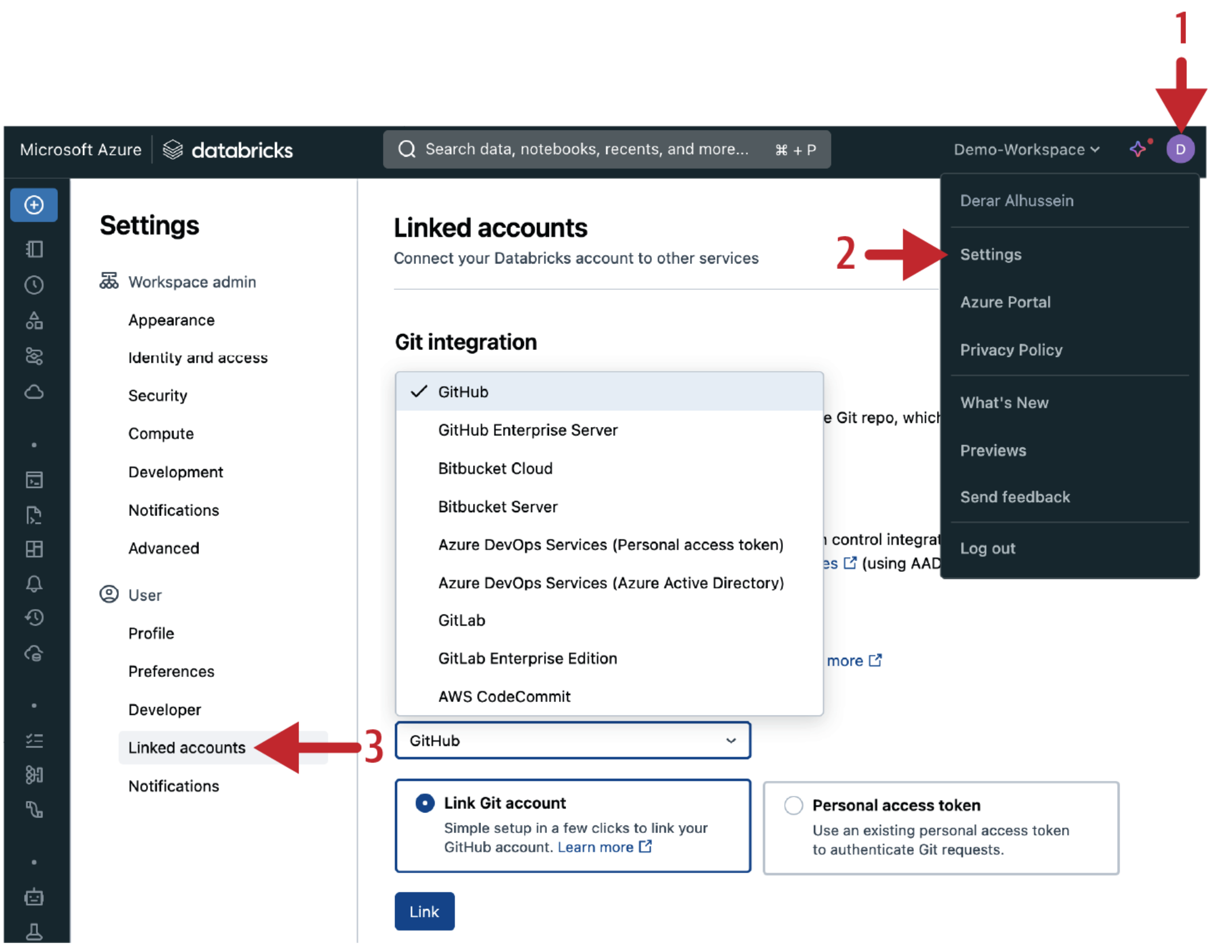


Figure 1-35. Git integration settings in Databricks

4. Install the Databricks app on GitHub: After authorizing the Databricks GitHub App, click Configure in GitHub, as shown in Figure 1-36, to configure the app installation on your GitHub account.

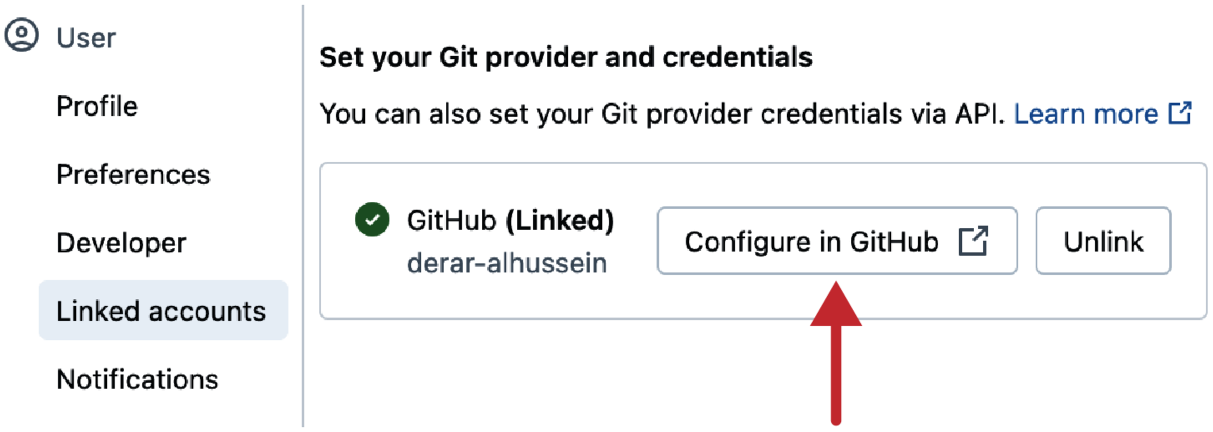
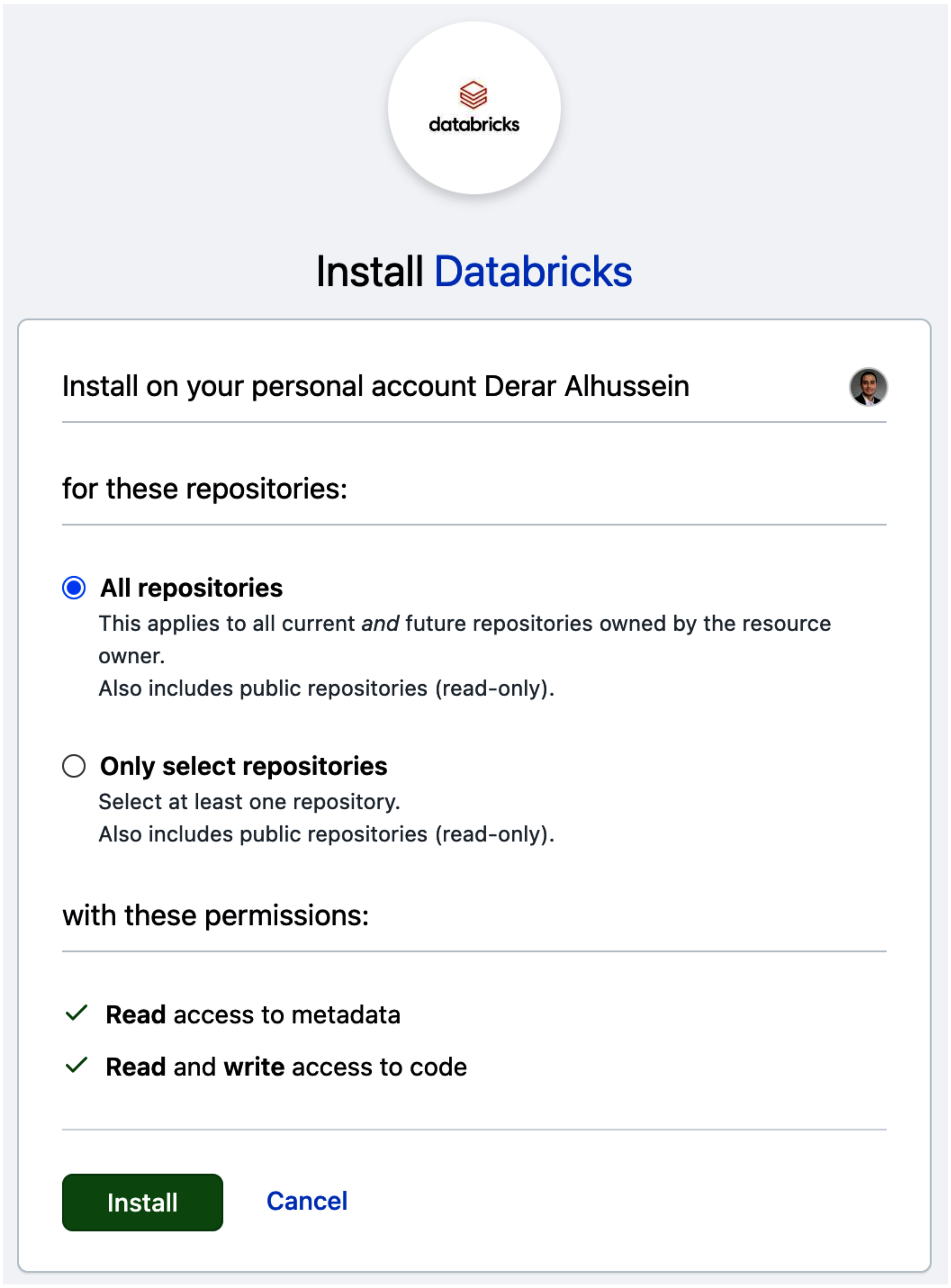


Figure 1-36. Configuring GitHub integration in Databricks

In the configuration page, you can choose to grant access to all your repositories or select specific repositories that Databricks can interact with, as displayed in Figure 1-37.

Once you confirm your selections, click Install to complete the setup. With this, your Databricks workspace is now fully integrated with your chosen Git provider, enabling seamless version control and collaboration.


Install Databricks

Install on your personal account Derar Alhussein

for these repositories:

[✓ All repositories]
This applies to all current *and* future repositories owned by the resource owner.
Also includes public repositories (read-only).

Only select repositories
Select at least one repository.
Also includes public repositories (read-only).

with these permissions:

✓ **Read** access to metadata
✓ **Read** and **write** access to code

Figure 1-37. Databricks app installation in GitHub account

Creating Git Folders

For effective collaboration and version control, integrating a private GitHub repository with Databricks is essential. This process involves creating a Git folder within your Databricks workspace and linking it to your private repository. The following is a step-by-step guide to help you achieve this integration:

1. Creating a private GitHub repository: To begin, ensure you have a private GitHub repository set up. If you haven’t done so yet, create a new one from your GitHub account and copy its URL.
2. Cloning the repository in Databricks: Now that you have your private repository ready, follow these steps to clone it into your Databricks workspace:
    1. Navigate to your workspace browser: In your Databricks workspace, navigate to the Workspace tab to access your Home directory.
    2. Create a Git folder: At the top of your directory, click the Create button and select “Git folder” from the drop-down menu, as illustrated in Figure 1-38.

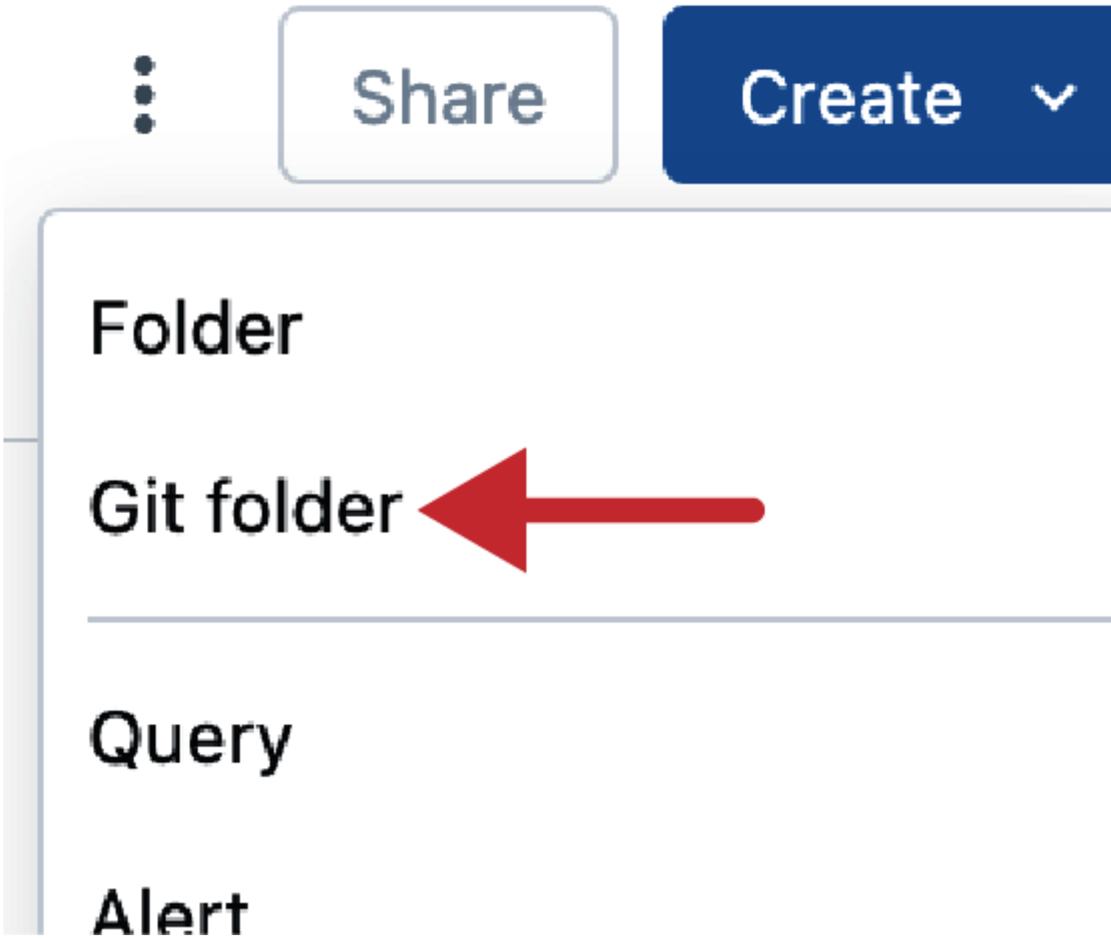


Figure 1-38. Adding a Git folder using the Create button in the workspace browser

This action will open a dialog box where you can specify the GitHub repository you want to clone, as shown in Figure 1-39.

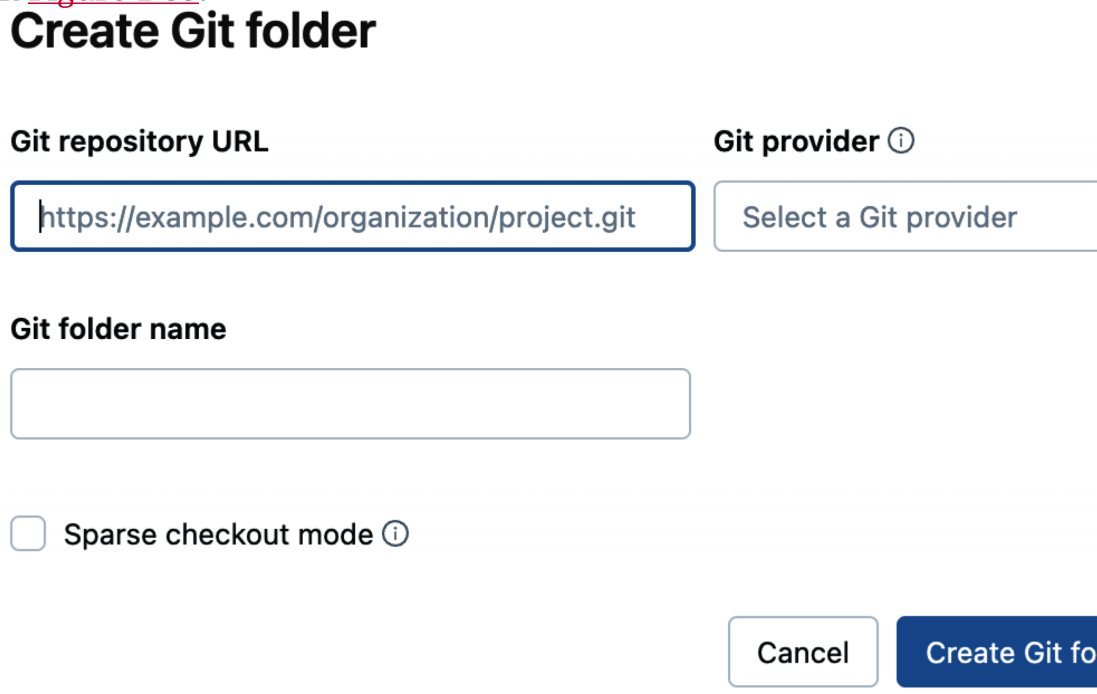
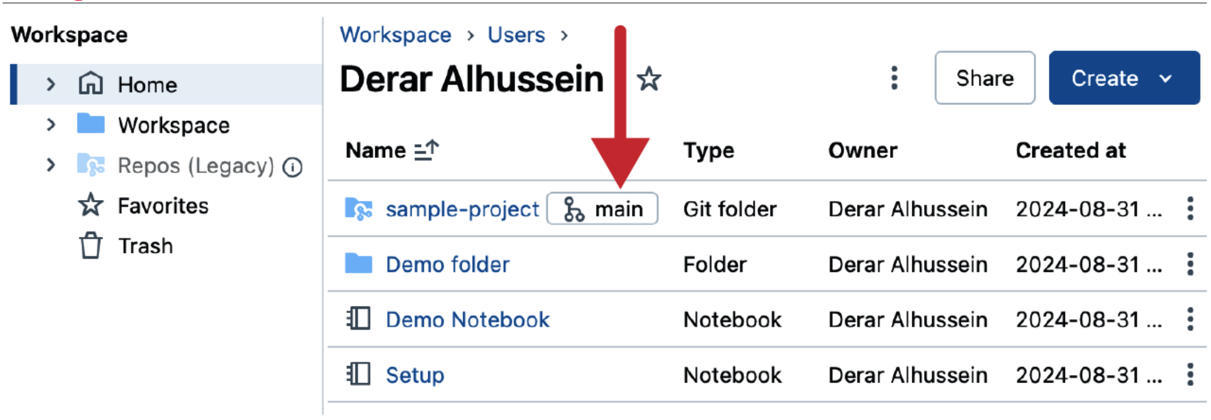


Figure 1-39. Git folder creation dialog

3. Paste the GitHub repository URL: In the Git folder creation dialog, paste the URL of your private repository. The interface will automatically detect the Git provider (e.g., GitHub) and fill in the repository name based on the URL provided.
4. Create the Git folder: After confirming the details, click the "Create Git folder" button to clone the repository.

Once the repository is cloned, you can navigate through its contents like any other folder. Git folders are easily recognized within the workspace browser by the current branch name displayed next to the folder name, as illustrated in Figure 1-40.



Name	Type	Owner	Created at
sample-project main	Git folder	Derar Alhussein	2024-08-31 ...
Demo folder	Folder	Derar Alhussein	2024-08-31 ...
Demo Notebook	Notebook	Derar Alhussein	2024-08-31 ...
Setup	Notebook	Derar Alhussein	2024-08-31 ...

Figure 1-40. Branch name indicator in Git folders

Managing Git Branches

Branches are a fundamental aspect of Git, allowing multiple developers to work on different features or fixes simultaneously without interfering with the main codebase.

By default, your Git repository will open on the main branch. To create a new local branch (e.g., a development branch), follow these steps:

1. Open the Git dialog: Click the branch name indicator next to the folder name. This will open the Git dialog, as shown in Figure 1-41.

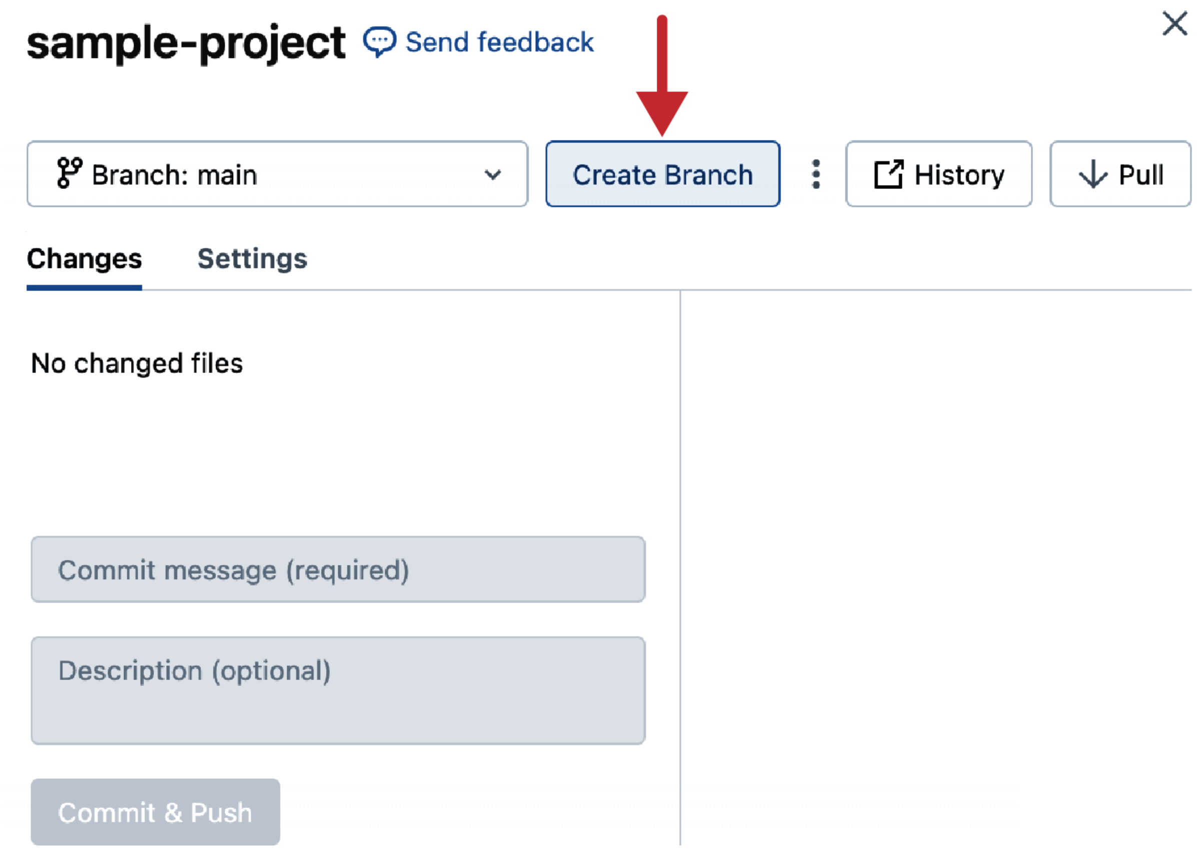


Figure 1-41. Git dialog in Databricks

2. Create the branch: Click the Create Branch button, specify the branch name (e.g., “dev” branch), and click Create.

This creates a local branch in the workspace and activates it immediately, allowing you to start working on your changes, just like you would on your machine. You can easily switch between branches at any time using the drop-down menu next to the Create Branch button.

With your development branch selected, you can begin working on your project by creating new notebooks or importing existing ones. Any edits you make to the source code are contained within this branch, which keeps the main branch stable and unaffected. Once your updates are ready and thoroughly tested, you can commit and push them to the remote repository.

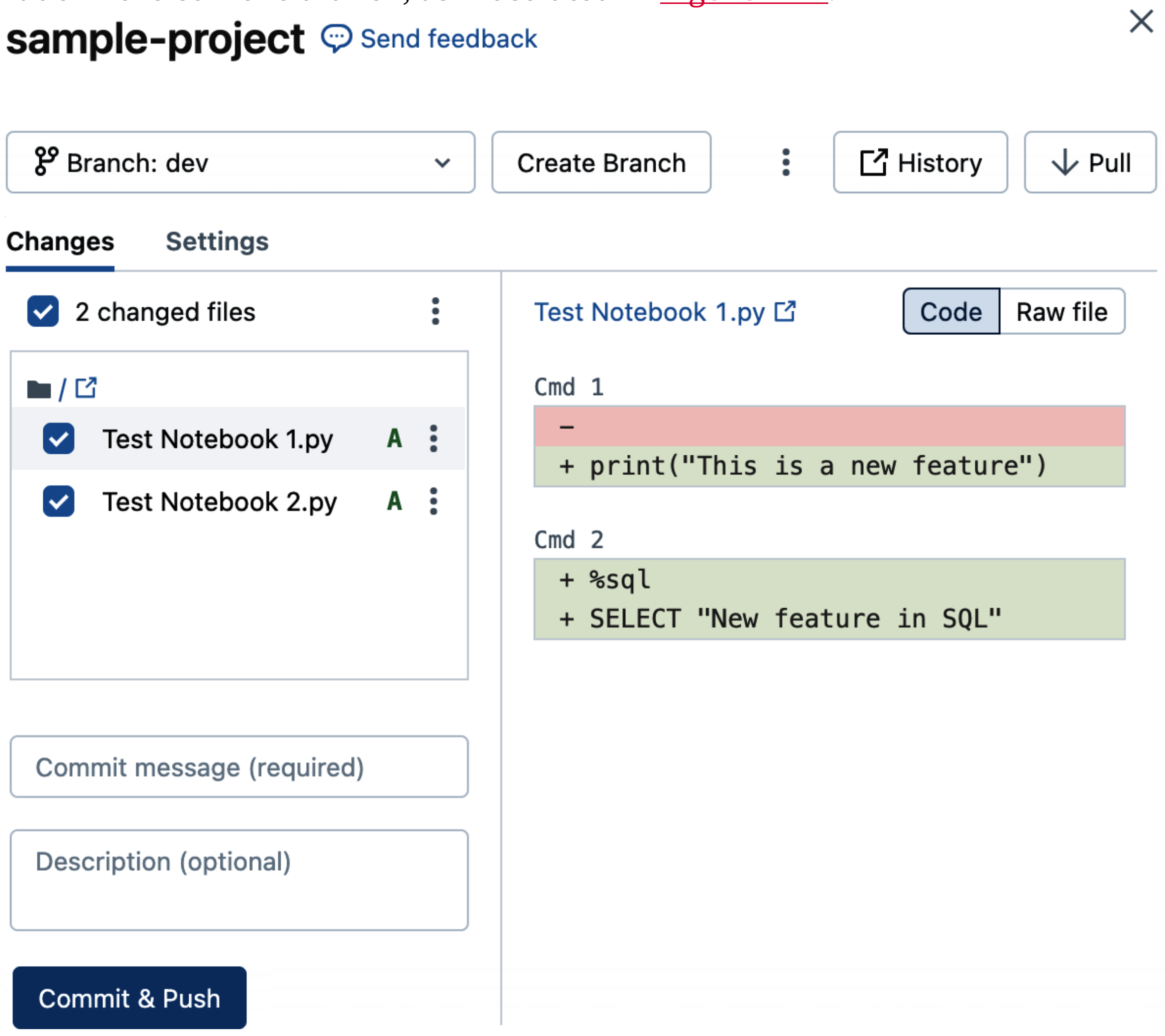
Committing and Pushing Changes

Once you’ve made changes in your Git folder, you can commit and push these changes to the remote repository to ensure your work is saved and shared with others.

To commit your changes, follow these steps:

1. Open the Git dialog: Click the branch name indicator to open the Git dialog.

2. Review changes: The Git dialog will display all the modifications made in the current branch, as illustrated in Figure 1-42.



```
Cmd 1
-
+ print("This is a new feature")

Cmd 2
+ %sql
+ SELECT "New feature in SQL"
```

Figure 1-42. Committing and pushing changes using the Git dialog

3. Add a commit message: At the bottom of the dialog, write a descriptive commit message summarizing the changes and any other information your organization might require.
4. Commit & Push: Click the blue Commit & Push button. This will save your changes locally and then push them to the remote repository on GitHub.

After pushing your changes, you can verify the update on the GitHub website. Navigate to your private repository and check the development branch to ensure the changes have been successfully applied.

Pulling Changes from GitHub

To keep your local repository in sync with the remote repository, especially after merging branches or when working in collaboration with others, pulling new changes is a common operation.

Synchronizing with merged pull requests

As a fundamental principle of Git, changes made in one branch are isolated from other branches until explicitly merged. To verify this, switch to the main branch in your Git folder by selecting it from the branch drop-down in the Git dialog. You will notice that the changes made in the dev branch are not visible in the main branch until a pull request (PR) is created and merged.

Databricks Git folders do not support creating pull requests directly; this must be done through your Git provider. For GitHub, follow these steps:

1. Create a pull request: On GitHub, navigate to your *sample-project* repository and create a pull request to merge changes from the dev branch into the main branch.
2. Merge the pull request: Once the pull request is reviewed and approved, merge it into the main branch on GitHub.

Pulling changes

To pull changes from the remote main branch to your local repository in Git folder, follow these steps:

1. Open the Git dialog: Click the branch name indicator in your Git folder to open the Git dialog.
2. Initiate a pull: With the main branch selected, click the Pull button, as displayed in Figure 1-43.

This action will fetch and merge changes from the remote main branch into your local copy. After merging, you can review the updates in your folder to ensure the integration was successful. As a best practice, perform regular pulls to minimize conflicts, especially in collaborative development environments where multiple contributors are working on the same codebase.

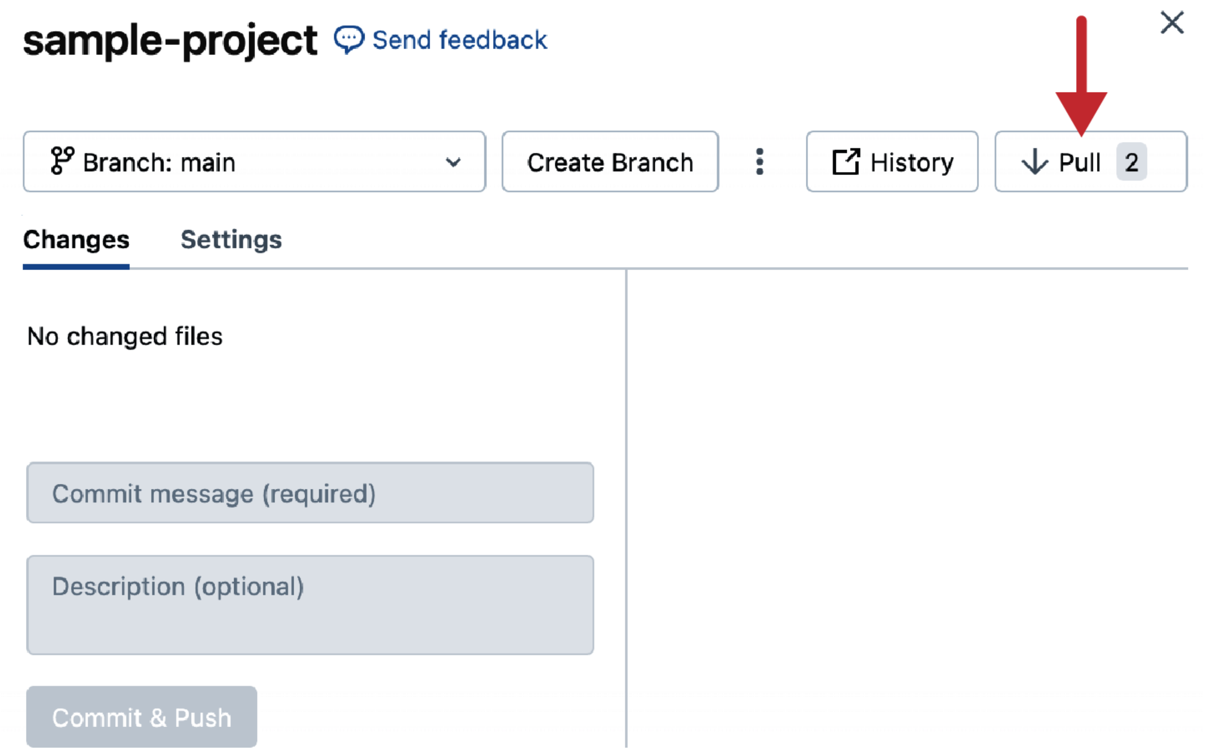


Figure 1-43. Pulling changes using the Git dialog

Conclusion

In conclusion, this chapter has provided an essential overview of the Databricks Data Intelligence Platform, covering its foundational architecture and offering practical guidance on working with clusters and notebooks. By mastering these essential components, you are now well-equipped to leverage the full potential of Databricks, which will be further explored in subsequent chapters.

Sample Exam Questions

Databricks certification exams primarily consist of multiple-choice questions with a single correct answer. The questions are categorized into two types: conceptual and code-based. Preparing for both types is critical for passing the certification exam.

Conceptual Questions

Conceptual questions focus on assessing your understanding of the core principles and features of Databricks. These questions typically ask you to recall definitions, describe functionality, or identify the role of different components within the Databricks environment.

Here is a sample conceptual question to give you a sense of what you might encounter:

Question 1. According to the Databricks lakehouse architecture, which of the following locations hosts the customer data?

1. Control plane
2. Databricks account
3. Customer's cloud account
4. Databricks Runtime
5. Workspace

This question tests your understanding of where Databricks stores customer data. The correct answer to this question is available in Appendix C.

Code-Based Questions

Code-based questions assess your ability to read, write, and debug code in the Databricks environment. These questions often present a block of code and ask you to either fill in missing portions of code, identify errors, or suggest modifications to ensure the code runs correctly.

Let's look at a sample code-based question:

Question 2. A data engineer has written the following code block within a cell in a SQL notebook, intending to list the files in the Databricks datasets directory:

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

print(files)
```

However, the code returns a syntax error.

What modification should be made to the code block to resolve this issue?

1. Replace `print(files)` with `display(files)`.
2. Use the following command instead:
   ```
   files = %fs ls 'databricks-datasets'
   print(files)
   ```
3. Use the following command instead:
   ```
   files = %fs ls 'databricks-datasets'
   display(files)
   ```

4. Add `%python` at the beginning of the cell.
5. Add `%run` at the beginning of the cell.

This question tests your ability to properly use the Databricks utility methods and understand how to display results in a notebook environment. The correct answer to this question can also be found in Appendix C.