

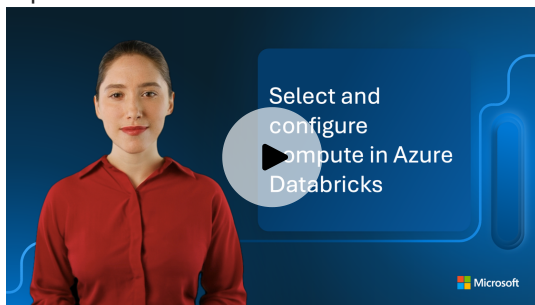
Select and Configure Compute in Azure Databricks

1. Introduction

<https://learn.microsoft.com/en-us/training/modules/select-and-configure-compute/1-introduction>

Introduction

Completed



Every Azure Databricks workload runs on compute resources, but choosing the wrong compute type or configuration leads to unnecessary costs, poor performance, or blocked functionality. **Serverless compute** starts in seconds but doesn't support RDD APIs. **Classic compute** offers complete flexibility but requires more management overhead. **SQL warehouses** excel at analytical queries while **job clusters** optimize for automated workflows. Understanding these differences helps you match compute to workload requirements.

Beyond selecting a compute type, configuration decisions shape how your workload performs. **Node types** determine processing capacity and memory availability. **Autoscaling** balances cost and responsiveness. **Access permissions** control who can use compute resources while **library installations** provide the dependencies your code needs. Each configuration choice affects multiple dimensions—performance, cost, security, and operational complexity.

Getting compute configuration right matters throughout the development lifecycle. During exploration, you need fast iteration cycles and minimal setup overhead. In production, stability and cost efficiency take priority. **Photon acceleration** can double query performance for SQL workloads. **Instance pools** reduce startup time but require paying for idle capacity. **Dedicated access modes** enable secure group collaboration while enforcing permission boundaries.

In this module, you learn to evaluate compute options systematically and configure resources that match your workload characteristics. You discover when serverless compute provides the best experience, how to tune performance settings for different scenarios, and best practices for managing access and dependencies across your organization.

2. Choose an appropriate compute type

<https://learn.microsoft.com/en-us/training/modules/select-and-configure-compute/2-choose-appropriate-compute-type>

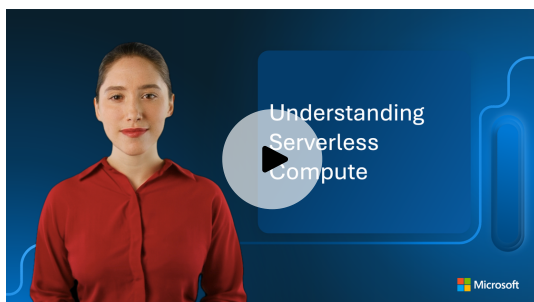
Choose an appropriate compute type

Completed

When you work with Azure Databricks, selecting the right **compute type** directly affects your costs, performance, and operational complexity. Each compute option serves different workload patterns and comes with distinct tradeoffs in startup time, scalability, and management overhead.

Understanding these options helps you match compute resources to your specific needs. You can optimize for fast development cycles, minimize costs for production jobs, or balance both for analytics workloads.

Serverless compute



Serverless compute is managed entirely by Azure Databricks. You don't provision or configure infrastructure—Azure Databricks automatically allocates and scales resources based on your workload demands. These resources run in **Databricks' Azure subscription, not yours**, which means no virtual machines or networking components appear in your subscription.

With serverless compute, startup typically takes 2-6 seconds. The platform scales up rapidly when query volume increases and scales down during idle periods to minimize costs. This eliminates the need to estimate capacity or manage cluster configurations.

Serverless compute requires **Unity Catalog** and is available for:

- **Notebooks:** Interactive Python and SQL development with automatic resource allocation
- **Jobs:** Automated workflows that run without infrastructure setup
- **Pipelines:** Lakeflow Spark Declarative Pipelines with on-demand scaling
- **SQL warehouses:** Optimized SQL query execution with intelligent workload management

Serverless works best for exploratory analysis, ETL pipelines, business intelligence workloads, and scenarios where startup latency matters. The versionless runtime means Azure Databricks automatically applies upgrades, so you always run on the latest features without migration effort.

However, serverless has limitations. You can't use **RDD APIs** (Resilient Distributed Dataset), **R language**, **JAR libraries** in notebooks, or custom Spark configurations. If your workload requires these capabilities, consider classic compute instead.

Classic compute



Classic compute gives you full control over cluster configuration. You create, size, and manage compute resources that run directly in **your Azure subscription**, giving you visibility and control over the underlying infrastructure.

Classic compute supports two access modes that determine how users interact with the cluster:

Standard access mode enables multiple users to share a single cluster concurrently. Lakeguard provides isolation between user code, preventing one user's work from interfering with another's. This mode works well for collaborative data engineering, shared analytics, and cost optimization through resource pooling.

Dedicated access mode assigns the cluster exclusively to a single user or group. With dedicated access, you get full machine-level privileges, which you need for RDD APIs, GPU workloads, R language support, or custom container environments.

Beyond access modes, classic compute offers different **cluster modes** that determine the cluster architecture:

Multi-node clusters consist of one driver node and one or more worker nodes. The driver coordinates execution while workers perform distributed computations in parallel. This architecture enables horizontal scaling—you can add worker nodes to process larger datasets or increase

parallelism. Multi-node clusters work well for production workloads that process large volumes of data or require high throughput.

Single-node clusters contain only a driver node with no worker nodes. All computation happens on the driver, which means workloads can't distribute across multiple machines. Single-node clusters suit specific scenarios like lightweight exploration, small dataset analysis, or machine learning experimentation where shuffle overhead would exceed the benefits of distribution. They're particularly useful for ML workflows with frameworks like scikit-learn that don't inherently distribute across nodes, or when testing notebooks with small data samples.

Keep in mind that single-node clusters have architectural limitations. They can't scale horizontally to handle increased load, and all processing relies on the resources of a single machine. For workloads requiring distributed processing, parallel task execution across large datasets, or fault tolerance through redundancy, use multi-node clusters instead.

Classic compute offers flexibility but requires more management. You configure instance types, autoscaling rules, and runtime versions. Startup time typically ranges from 3-7 minutes depending on cluster size. Unlike serverless, you select and manage the Databricks Runtime version yourself—choosing when to upgrade from one version to another (for example, from Runtime 13.3 LTS to 14.3 LTS). While underlying OS and security updates can be automated, runtime version changes require manual selection.

This compute type fits workloads that need features unavailable in serverless, require precise control over infrastructure, or have compliance requirements for resource isolation.

SQL warehouses



SQL warehouses are compute resources optimized specifically for SQL queries, analytics, and business intelligence. They come in three types, each with different performance characteristics.

Serverless SQL warehouses offer optimal performance and cost efficiency. They start in 2-6 seconds, use Intelligent Workload Management to predict query resource needs, and scale clusters dynamically based on demand. Photon and Predictive IO accelerate query execution. Choose serverless SQL warehouses for most SQL workloads—BI dashboards, ETL jobs, and ad hoc analysis.

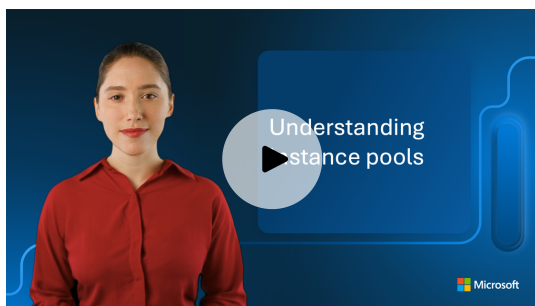
Pro SQL warehouses support Photon and Predictive IO but lack Intelligent Workload Management. They take approximately 4 minutes to start and scale less dynamically than serverless. You use pro

warehouses when you need custom networking configurations—for example, connecting to on-premises databases through federation or integrating with services in your virtual network.

Classic SQL warehouses offer entry-level SQL performance with Photon support only. They start in about 4 minutes and include basic autoscaling. Choose classic warehouses only when serverless and pro options aren't available or for basic interactive exploration with minimal performance requirements.

All SQL warehouse types optimize for SQL execution patterns, but serverless offers the most responsive scaling and lowest operational overhead.

Instance pools



Instance pools maintain a set of idle virtual machine instances ready for immediate use. When you create a cluster from a pool, startup time decreases because Databricks allocates instances from the pool instead of requesting new ones from Azure.

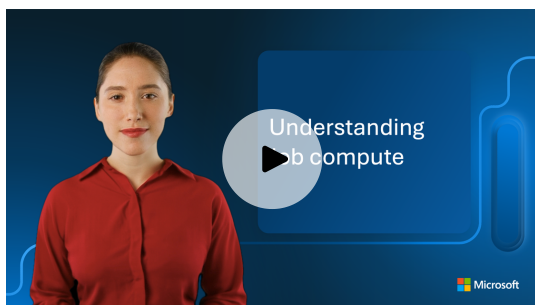
Pools reduce startup time from minutes to under a minute in many cases. You configure the minimum number of idle instances to keep warm and the maximum pool capacity. When clusters release instances, those instances return to the pool for reuse.

You pay for virtual machine costs while instances sit idle in the pool, but not for Azure Databricks compute units. This makes pools cost-effective when you run workloads frequently enough that the reduced startup time justifies the idle infrastructure cost.

With serverless compute available, pools matter less for most scenarios. Serverless starts faster and scales more efficiently without requiring you to maintain idle capacity. However, pools remain useful when you need classic compute features and want to optimize startup time for frequently run workloads.

Configure pools with spot instances for worker nodes to reduce costs, but use on-demand instances for driver nodes to maintain reliability.

Job compute



Job compute refers to clusters optimized for automated workflows rather than interactive development. You configure job compute through cluster policies that enforce best practices for production workloads.

Job clusters terminate automatically after completing their tasks, preventing unnecessary costs from idle resources. When you configure a job, you choose between serverless and classic job compute.

- **Serverless job compute** offers faster startup, automatic infrastructure management, and lower costs for most automated workloads.
- **Classic job compute** provides more configuration options for workloads that need features serverless doesn't support.

With classic job compute, you can use optimized settings like autoscaling and **spot instances**. **Spot instances** (also called Azure Spot VMs) use Azure's excess capacity at significantly reduced costs—often up to 90% cheaper than regular on-demand instances. Azure can reclaim these instances with only 30 seconds notice when it needs the capacity back. Spark's built-in fault tolerance automatically handles these interruptions by retrying failed tasks on other available nodes, which makes spot instances a viable cost-saving option for many batch processing and ETL workloads.

The Job Compute policy in Azure Databricks offers a template for creating production-ready job clusters with sensible defaults. It enforces the latest LTS (Long Term Support) runtime version and other reliability settings.

Compare compute types

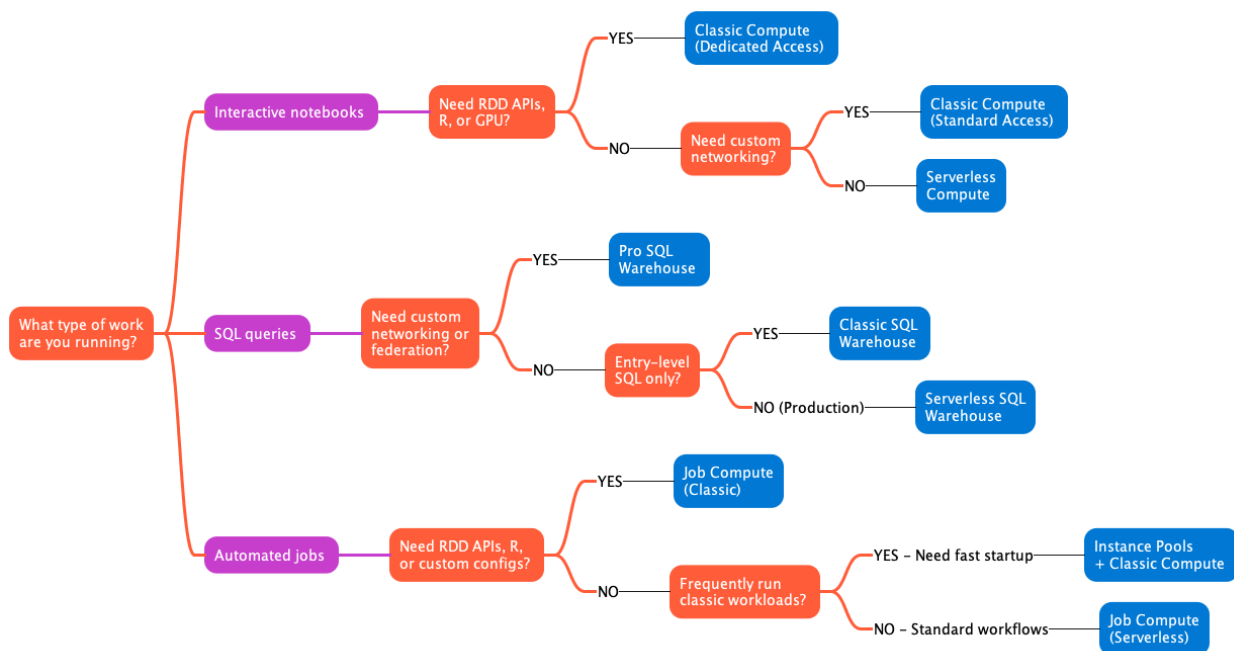


Different compute types suit different scenarios. The following table compares key characteristics to help you make informed decisions:

Compute type	Recommended for	Startup time	Management overhead	Cost efficiency	Key limitation
Serverless compute	Interactive development, ETL jobs, BI workloads	⚡ 2-6 seconds	🔧 Minimal - fully managed	🟢 High - scales to zero, pay only for usage	No RDD APIs, R, or JAR libraries
Classic compute (Standard)	Collaborative data engineering, shared analytics	🕒 3-7 minutes	🔧 Moderate - configure and monitor	🟡 Moderate - multi-user sharing reduces per-user cost	Requires Unity Catalog for governance
Classic compute (Dedicated)	RDD workloads, GPU jobs, R language, custom containers	🕒 3-7 minutes	🔧 Moderate - configure and monitor	🔴 Lower - single user/group only	Higher cost than shared resources
Instance pools	Frequently run classic workloads needing fast startup	🚀 <1 minute	⚠️ Higher - maintain idle capacity	🔄 Variable - justify idle cost with usage frequency	Pay for idle instances
SQL warehouse (Serverless)	SQL analytics, BI dashboards, reporting	⚡ 2-6 seconds	🔧 Minimal - intelligent workload management	🟢 High - dynamic scaling with Photon + Predictive IO	SQL workloads only
SQL warehouse (Pro)	SQL with custom networking, federation	🕒 ~4 minutes	🔧 Moderate - manual scaling configuration	🟡 Moderate - Photon + Predictive IO	Slower scaling than serverless
SQL warehouse (Classic)	Entry-level SQL exploration	🕒 ~4 minutes	🔧 Moderate - manual scaling configuration	🔴 Lower - basic Photon only	Limited performance features
Job compute (Serverless)	Automated workflows, production ETL	⚡ 2-6 seconds	🔧 Minimal - auto-terminated after completion	🟢 High - no idle costs	Same as serverless compute
Job compute (Classic)	Jobs requiring custom configurations	🕒 3-7 minutes	🔧 Moderate - configure policies	🟡 Moderate - auto-termination prevents idle waste	Requires infrastructure management

Choose the right compute type

Start your decision-making process by identifying your workload characteristics. The following diagram illustrates a decision flow to help you select the appropriate compute type:



Consider these questions:

What type of work are you running? If you're writing interactive notebooks, serverless compute offers the fastest iteration cycle. For SQL queries and BI dashboards, serverless SQL warehouses deliver optimal performance. Automated production jobs work well with serverless job compute unless you need custom configurations.

Does your code use specific APIs or languages? RDD APIs, R language, or GPU acceleration require classic compute with dedicated access mode. Python, SQL, and Scala workloads run on either serverless or standard classic compute.

How frequently does this workload run? Infrequent workloads benefit most from serverless because you pay only during execution. Recurring workloads might justify instance pools if you use classic compute, though serverless often still provides better economics.

Do you need custom networking or specific infrastructure? Custom virtual networks, on-premises connectivity, or specific instance types require classic compute or pro SQL warehouses. Serverless operates in the Databricks-managed subscription without custom network integration, so resources don't appear in your Azure subscription.

What are your performance requirements? For latency-sensitive workloads, serverless compute starts 2-4x faster than classic options. For predictable performance at scale, both serverless and classic can meet requirements, but serverless adapts more dynamically to load variations.

For most scenarios, start with serverless options. They minimize operational overhead, optimize costs through automatic scaling, and provide the fastest development experience. Switch to classic compute only when you encounter a specific limitation that serverless doesn't support.

3. Configure compute performance

<https://learn.microsoft.com/en-us/training/modules/select-and-configure-compute/3-configure-compute-performance>

Configure compute performance

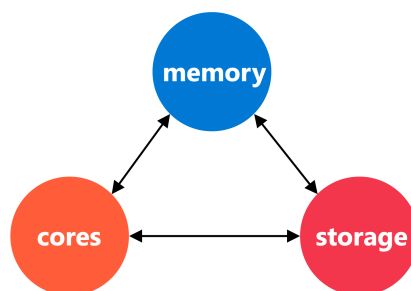
Completed

Configuring compute resources involves balancing performance requirements with cost considerations. Over-provisioning leads to unnecessary expenses, while under-provisioning can cause stability issues and slow query execution. Understanding how to configure compute settings helps you optimize resources for your workload.

Understand compute resource components



Compute performance depends on three key factors working together. Each factor influences how efficiently your workload runs and how much it costs.



Total executor cores determine the maximum parallelism available for processing data. More cores allow Spark to process more tasks simultaneously. A cluster with 8 workers, each having 4 cores, provides 32 cores total for parallel processing.

Total executor memory affects how much data can be processed in memory before spilling to disk. Memory-intensive operations like joins and aggregations benefit from larger memory configurations. When memory runs out, Spark writes data to disk, which significantly slows performance.

Local storage provides temporary space for shuffle operations and caching. During shuffle operations, Spark writes intermediate data to local disks on worker nodes. Fast local storage reduces the time spent on these operations.

With this understanding of compute components, you can make informed decisions about node types and cluster size.

Configure node types and cluster size



Node type selection directly impacts both performance and cost. Different instance families serve different workload characteristics.

Memory-optimized instances work well for workloads with large **joins**, **aggregations**, or data that needs to stay in memory. These instances provide more RAM per core, reducing the likelihood of spilling data to disk. Examples include **E-series** VMs, which offer high memory-to-core ratios ideal for in-memory analytics.

Compute-optimized instances suit workloads that perform complex **calculations** but don't require large amounts of memory. ETL jobs with straightforward transformations often run efficiently on these instances. Examples include **F-series** VMs, which provide high CPU performance with lower memory ratios.

Storage-optimized instances benefit workloads that repeatedly read the same data or require fast **local disk access**. Data analysis workloads with caching enabled perform better with these instances. Examples include **L-series** VMs, which offer fast local **NVMe storage** for high I/O workloads.

GPU-accelerated instances provide graphics processing units designed for computationally intensive workloads like **machine learning**, **deep learning**, and **image processing**. These instances can accelerate model training by 10-100x compared to CPU-only clusters. Examples include **NC-series** and **ND-series** VMs with NVIDIA GPUs. GPU instances require **Databricks Runtime ML** and work best for tasks like training neural networks, fine-tuning large language models, or running inference on complex models.

Performance

Databricks runtime version ⓘ

Runtime: 17.3 LTS (Scala 2.13, Spark 4.0.0) ▼

☒ Use Photon Acceleration ⓘ

Worker type ⓘ	Min workers	Max workers	☒ Spot instances ⓘ
Standard_DC4as_v5 16 GB Memory, 4 Cores ▼	2	8	

Driver type

Same as worker 16 GB Memory, 4 Cores ▼

☒ Enable autoscaling ⓘ

☒ Terminate after 120 minutes of inactivity ⓘ

The balance between number of workers and instance size affects performance differently depending on your workload. Two workers with 16 cores and 128 GB RAM each provide the same total compute and memory as eight workers with 4 cores and 32 GB RAM each. However, the configuration with **fewer, larger workers** reduces network traffic during shuffle operations, while **more smaller workers** can provide better parallelism for highly distributed workloads.

For analytical workloads with many shuffle operations, **fewer larger workers** typically perform better. For simple batch processing that benefits from high parallelism, **more smaller workers** might be more cost-effective.

Configure autoscaling

Autoscaling adjusts the number of workers based on workload demands, helping you maintain performance while controlling costs.

When you enable autoscaling, you set minimum and maximum worker counts. Azure Databricks monitors workload requirements and adds workers when needed, up to the maximum you specified. When demand decreases, workers are removed down to the minimum.

Azure Databricks uses **optimized autoscaling** by default when you enable autoscaling. Optimized autoscaling scales up quickly in two steps from minimum to maximum. It can scale down even when the cluster isn't idle by monitoring shuffle file state. For **job compute**, it evaluates utilization every 40 seconds. For **all-purpose compute**, it checks every 150 seconds.

Consider autoscaling for workloads with variable resource needs throughout execution. Data exploration sessions often start with small data samples and later process larger datasets. Autoscaling adds workers when you process the larger datasets and removes them when you return to smaller samples.

For predictable workloads that maintain consistent resource usage, a fixed number of workers often provides more stable performance and simpler capacity planning. The overhead of scaling decisions can slightly impact performance for steady-state workloads.

Autoscaling works particularly well with instance pools. Set your minimum workers equal to or less than the minimum idle instances in the pool. This ensures quick scaling because the instances are already provisioned and ready.

Configure termination settings

Automatic termination prevents idle compute resources from accumulating unnecessary costs while maintaining availability for scheduled workloads.



When you configure automatic termination, you specify an inactivity period in minutes. If no commands run on the cluster for longer than this period, Azure Databricks terminates the cluster. The cluster configuration remains available for restart when needed.

For interactive workloads like data analysis, set the termination period based on typical session patterns. A 45-minute timeout works well for most use cases, giving data engineers time to review results between queries without leaving clusters idle for hours.

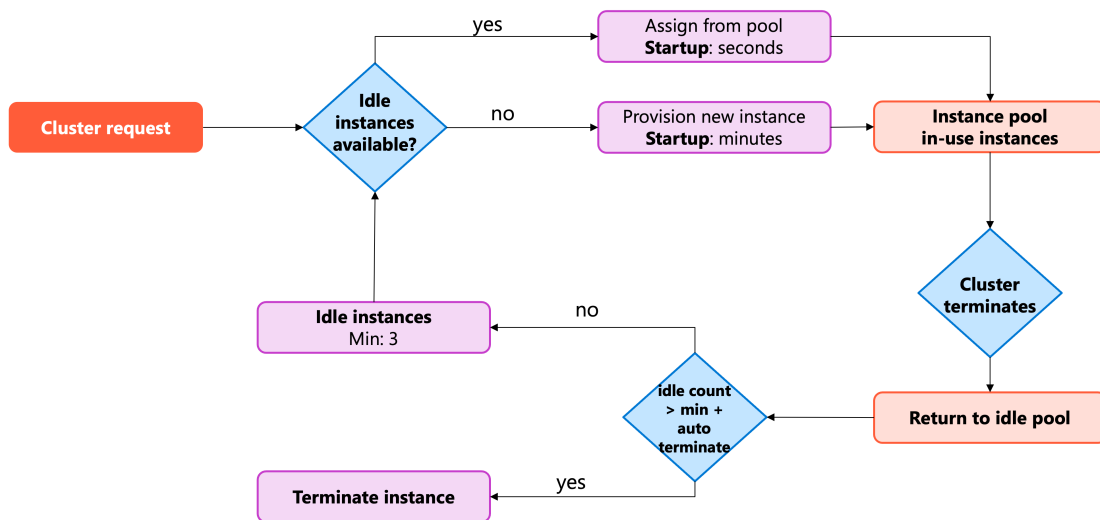
For job compute, automatic termination happens after the job completes. The cluster starts automatically when the next scheduled run begins, so you don't need to manage startup manually.

Spot instances reduce costs but come with availability trade-offs. Azure can reclaim spot instances when capacity is needed elsewhere. For **worker nodes**, spot instances work well because Azure Databricks/Spark can handle worker failures. However, always use **on-demand instances** for **driver nodes**. If the driver is reclaimed, the entire cluster fails.

Enable **decommissioning** when using spot instances to reduce task failures. When a spot instance receives a preemption notice, decommissioning migrates shuffle and cached data to healthy workers before the instance terminates. This reduces the need to recompute lost data.

Use instance pools

Instance pools maintain a set of idle instances ready for immediate use, reducing cluster startup time from minutes to seconds.



Configure the **minimum idle instances** to match your typical concurrent cluster needs. If you regularly run three notebooks simultaneously, maintain at least three idle instances. These instances remain available even when not in use, providing instant cluster startup.

Create Pool

Name

Min idle ⓘ

Max capacity ⓘ

Idle Instance Auto Termination ⓘ

Terminate instances above minimum after minutes of idle time.

Instance Type ⓘ

Standard_DC4as_v5

16 GB Memory, 4 Cores

▼

Preloaded Databricks Runtime Version

None

▼

☐ Use Photon preloaded runtimes ⓘ

Instances Tags

On-demand/Spot

☒ All On-demand ☐ All Spot

Set **maximum capacity** to control costs and prevent one workload from consuming all available resources. When multiple teams share a workspace, pools with maximum capacity settings ensure fair resource distribution. For example, with a 100-instance quota, you might create two pools each with a 50-instance maximum for two teams.

The **idle instance auto termination** setting removes instances that exceed your minimum idle count after the specified period. If you set minimum idle to 3 and auto termination to 30 minutes, a pool that scales up to 8 instances will reduce back to 3 instances after 30 minutes of inactivity.

Preloading a Databricks Runtime version on pool instances accelerates cluster launches even further. When creating a cluster, if you select the preloaded runtime, the cluster starts almost immediately because the runtime is already installed on idle instances.

Pools work best for workloads with frequent cluster creation and termination cycles. Development teams creating and destroying clusters throughout the day see significant time savings. Production jobs that run on dedicated long-running clusters don't benefit as much from pools.

Balance cost and performance

Achieving the right balance between cost and performance requires understanding your workload characteristics and adjusting configurations accordingly.

Start with conservative settings and monitor performance. If you see frequent spilling to disk or slow query execution, increase memory or core count. If utilization remains low, reduce cluster size or enable autoscaling.

Note

Use the **Spark UI** to identify performance issues. Check the **Jobs Timeline** to find long-running stages, and view the **stage details page** for spill statistics showing **Shuffle Spill (Memory)** and **Shuffle Spill (Disk)**. Compare stage durations to identify bottlenecks and slow queries.

Use serverless compute when your workload supports it. Serverless eliminates configuration decisions and automatically scales based on demand, often providing the best cost-performance balance without manual tuning.

Regular monitoring helps you identify optimization opportunities. Review cluster metrics to see actual utilization compared to provisioned capacity. Adjust node types, worker counts, or scaling settings based on observed patterns rather than assumptions.

Note

Monitoring and observability are covered in detail in a later module.

4. Configure compute features

Configure compute features

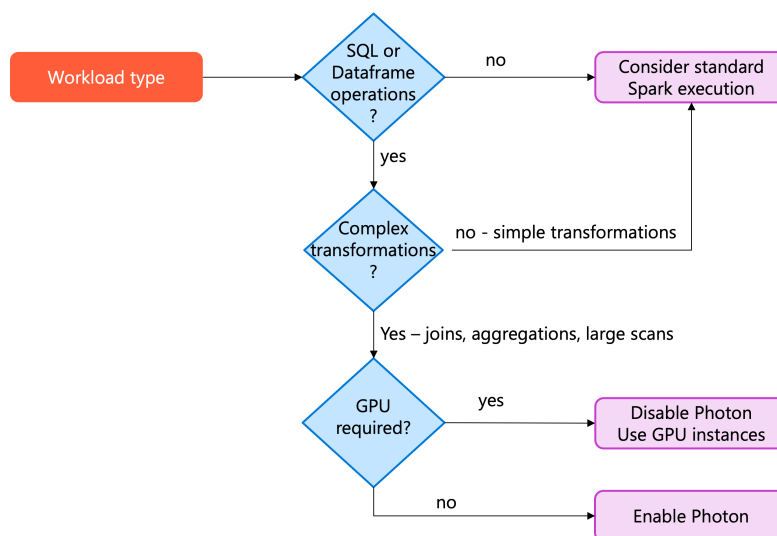
Completed

Compute features determine the functional capabilities available to your workloads. Unlike performance settings that tune resource allocation, feature settings enable specific technologies and runtime environments. Selecting appropriate features ensures your compute resource supports your workload requirements while maximizing efficiency.

Enable Photon acceleration



Photon is a query execution engine that replaces traditional Spark components with optimized native code. When you enable Photon, your compute resource uses this accelerated engine for SQL queries and DataFrame operations.



With Photon enabled, queries that involve **complex transformations** run faster. Operations like **joins**, **aggregations**, and **scans** across large tables benefit most from Photon's optimization.

Workloads that frequently access disk, process wide tables, or repeatedly transform data also see significant performance gains. For example, a data analyst running hourly aggregation queries across millions of rows will experience faster query completion times with Photon enabled.

Performance

Databricks runtime version ⓘ

Runtime: 17.3 LTS (Scala 2.13, Spark 4.0.0)



☒ Use Photon Acceleration ⓘ

Without Photon, the same workloads rely on standard Spark execution, which may suffice for simple transformations but lacks the performance optimizations Photon provides. Simple batch ETL jobs that process small datasets or complete in under two seconds typically see minimal benefit from Photon. In these cases, the overhead of enabling Photon may not justify the additional compute cost.

Photon is enabled by default on Databricks Runtime 9.1 LTS and above. You can verify or change this setting during compute creation under the Performance section. Keep in mind that **Photon isn't supported on GPU-enabled clusters**. If your workload requires GPU instances for machine learning or deep learning tasks, you must disable Photon.

Select Databricks Runtime and Spark version

Databricks Runtime provides the core components that run on your compute resource, including Apache Spark and additional optimizations. The runtime version you select determines which features are available and how your code interacts with preloaded packages.

Runtime selection impacts your workload in different ways depending on your use case. For **all-purpose compute** used in interactive development, the most current runtime version ensures you have the latest optimizations and compatibility with modern packages. This approach works well when you're actively developing notebooks and exploring data, as you benefit from continuous improvements.

For **job compute** running operational workloads, stability becomes more important than having the newest features. **Long Term Support (LTS)** runtime versions provide extended compatibility, allowing you to thoroughly test your workload before upgrading. A data engineering pipeline that runs daily transformations benefits from this stability, as unexpected runtime changes won't disrupt production processes.

Databricks runtime version ⓘ

Runtime: 17.3 LTS (Scala 2.13, Spark 4.0.0) ▼		
Standard	>	17.3 LTS Scala 2.13, Spark 4.0.0
ML	>	17.2 Scala 2.13, Spark 4.0.0
		17.1 Scala 2.13, Spark 4.0.0
		17.0 Scala 2.13, Spark 4.0.0
		16.4 LTS (Scala 2.13) Scala 2.13, Spark 3.5.2
		16.4 LTS (Scala 2.12) Scala 2.12, Spark 3.5.2
		15.4 LTS Scala 2.12, Spark 3.5.0
		14.3 LTS Scala 2.12, Spark 3.5.0
		13.3 LTS Scala 2.12, Spark 3.4.1
		12.2 LTS Scala 2.12, Spark 3.3.2
		11.3 LTS Scala 2.12, Spark 3.3.0
		10.4 LTS Scala 2.12, Spark 3.2.1

The **Spark version** is tied to your selected runtime, so choosing a runtime automatically determines which version of Spark you're using. Newer runtimes include more recent Spark versions with additional features and bug fixes. Older runtimes provide stability but may lack features introduced in recent Spark releases. When planning upgrades, consider testing your workload on the new runtime version in a development environment before applying changes to production compute resources.

Configure machine learning environments

Machine learning workloads require specialized runtime environments and hardware configurations. **Databricks Runtime ML** includes pre-installed machine learning libraries, GPU drivers, and frameworks like CUDA that support deep learning tasks.

When you configure compute for machine learning, start by selecting a **Databricks Runtime ML** version instead of the standard runtime. This runtime comes with popular libraries already installed, reducing setup time and ensuring compatibility between packages. For initial model experimentation, a **single-node** compute resource with a large instance type provides sufficient resources while minimizing shuffle overhead.

Databricks runtime version ⓘ

Runtime: 17.3 LTS ML (GPU, Scala 2.13, Spark 4.0.0) ▼		NVIDIA EULA ⓘ	
Standard >	17.3 LTS ML	GPU, Scala 2.13, Spark 4.0.0	
ML >	17.3 LTS ML	Scala 2.13, Spark 4.0.0	
	17.2 ML	GPU, Scala 2.13, Spark 4.0.0	
	17.2 ML	Scala 2.13, Spark 4.0.0	
	17.1 ML	GPU, Scala 2.13, Spark 4.0.0	
	17.1 ML	Scala 2.13, Spark 4.0.0	
	17.0 ML	GPU, Scala 2.13, Spark 4.0.0	
	17.0 ML	Scala 2.13, Spark 4.0.0	
	16.4 LTS ML (Scala 2.13)	GPU, Scala 2.13, Spark 3.5.2	
	16.4 LTS ML (Scala 2.12)	GPU, Scala 2.12, Spark 3.5.2	
	16.4 LTS ML (Scala 2.13)	Scala 2.13, Spark 3.5.2	
	16.4 LTS ML (Scala 2.12)	Scala 2.12, Spark 3.5.2	

GPU instances enable training of deep learning models that would be impractical on CPU-only compute. Models involving image recognition, natural language processing, or neural networks benefit significantly from GPU acceleration. With GPU-enabled instances, operations that would take hours on CPUs complete in minutes. Keep in mind that **Photon must be disabled when using GPU instances**, as these features aren't compatible.

5. Install libraries for compute

<https://learn.microsoft.com/en-us/training/modules/select-and-configure-compute/5-install-libraries-for-compute>

Install libraries for compute

Completed

When you run notebooks and jobs on Azure Databricks compute, you often need third-party packages or custom code that isn't included in the default runtime. Installing **libraries** at the **cluster level** ensures that every notebook and job using that compute has access to the same dependencies, creating a consistent execution environment.

Understanding how to install libraries effectively becomes critical as your data engineering workflows grow in complexity. You need to know which installation method to use, where to store library files, and how access modes affect your options.



Understand compute-scoped libraries

Compute-scoped libraries install on a cluster and become available to all notebooks and jobs that run on that cluster. Unlike **notebook-scoped libraries** that install only for a specific notebook session, compute-scoped libraries persist across cluster restarts and provide a shared environment for all users.

When you install a library at the cluster level, Azure Databricks automatically reinstalls it every time the cluster starts. This behavior ensures consistency—you don't need to manually reinstall dependencies after stopping and restarting a cluster. All notebooks attached to the cluster can import and use the installed packages immediately.

Note

To install libraries on a cluster, you must have **CAN MANAGE** permission on that cluster. This permission allows you to modify cluster configuration, including adding and removing libraries. Without this permission, you won't be able to access the library installation interface.

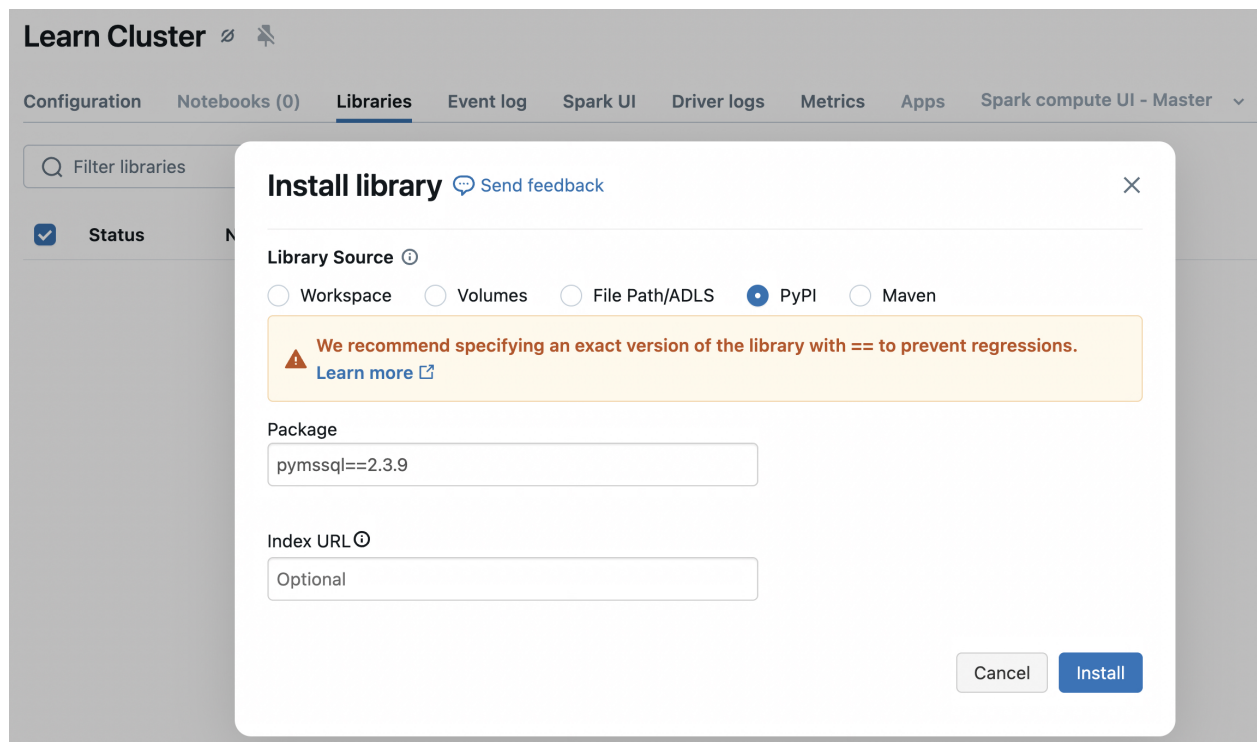
Compute-scoped libraries support **Python wheels**, **Java JAR files**, and **R packages**. You can install them from package repositories like **PyPI** and **Maven**, or from files stored in **workspace files**, **Unity Catalog volumes**, or cloud object storage. The installation method you choose depends on your library type, cluster access mode, and organizational security requirements.

However, compute-scoped libraries have an important limitation: any library you install affects every notebook on the cluster. If different teams need conflicting versions of the same library, you'll need separate clusters or notebook-scoped installations to avoid conflicts.

Install libraries from package repositories

Package repositories provide the most common way to install libraries. PyPI hosts Python packages, Maven hosts Java and Scala libraries, and **CRAN** hosts R packages. These repositories automatically handle **dependency resolution** and version management.

To install a library from PyPI, select **PyPI** as the library source and enter the package name. For production workloads, specify an exact version to ensure reproducibility: `pymssql==2.3.9`. Without a version number, Azure Databricks installs the latest available version, which might change between installations and potentially break your code.



The screenshot shows the 'Learn Cluster' interface with the 'Libraries' tab selected. A modal dialog titled 'Install library' is open. It features a 'Library Source' section with radio buttons for 'Workspace', 'Volumes', 'File Path/ADLS', 'PyPI' (selected), and 'Maven'. Below this is a warning message: 'We recommend specifying an exact version of the library with == to prevent regressions.' with a 'Learn more' link. The 'Package' input field contains 'pymssql==2.3.9'. The 'Index URL' field is optional and contains 'Optional'. At the bottom right are 'Cancel' and 'Install' buttons.

Maven libraries require **coordinates** in the format `groupId:artifactId:version`. For example, to install the Microsoft JDBC Driver for SQL Server library, you would use `com.microsoft.sqlserver:mssql-jdbc:13.2.1.jre11`. You can search for packages directly in the installation dialog if you don't know the exact coordinates. Maven also supports excluding specific transitive dependencies that might conflict with other installed libraries.

Learn Cluster

Configuration Notebooks (0) Libraries Event log Spark UI Driver logs Metrics Apps Spark compute UI - Master

Filter libraries

Install library Send feedback

Library Source

☐ Workspace ☐ Volumes ☐ File Path/ADLS ☐ PyPI ☒ Maven

Coordinates

com.microsoft.sqlserver:mssql-jdbc:13.2.1.jre11 Search Packages

Repository

Optional

Exclusions

Dependencies to exclude (log4j:log4j,junit:junit)

Cancel Install

For R packages from CRAN, provide the package name. Unlike Python and Java libraries, CRAN installations always pull the latest version from the configured mirror. To pin specific R package versions, you need to store the package files in workspace files or volumes instead of installing from CRAN.

With clusters configured in **standard access mode**, Maven coordinates and JAR file paths require **allowlist approval** before installation. This security measure ensures admins review and approve libraries that run on shared compute resources.

Note

To learn more about configuring and managing **allowlists** for libraries, see the [documentation](#).

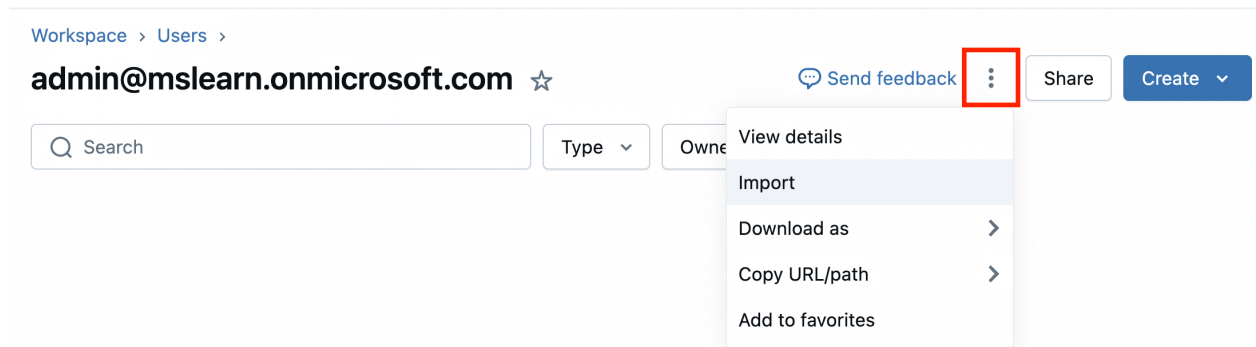
Install libraries from files

Storing library files in **workspace files** or **Unity Catalog volumes** gives you precise control over which library versions your clusters use. This approach works well when you need libraries not available in public repositories, custom packages you've built internally, or specific versions no longer available from package repositories.

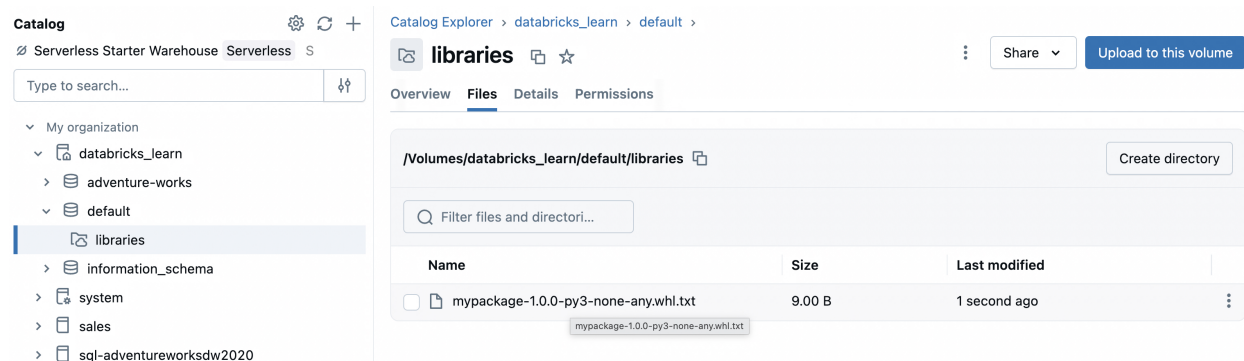
Using workspace files and Unity Catalog volumes for library installation maintains centralized management rather than bypassing security controls with ad-hoc installations like direct pip3 commands or unmanaged custom scripts executed from notebooks. Unity Catalog volumes provide

enhanced governance through Unity Catalog's access control model, ensuring that all library installations are tracked with audit logs and protected by fine-grained permissions.

Workspace files provide a convenient location for library storage with a 500 MB file size limit. To install a library from workspace files, upload your wheel, JAR, or requirements.txt file through the workspace Import dialog, then reference it during library installation using a path like `/Workspace/Users/you@example.com/libraries/mypackage-1.0.0-py3-none-any.whl`.



Unity Catalog volumes offer enhanced security and governance for library storage. You control access through Unity Catalog permissions, ensuring only authorized users can read or modify library files. Upload files to a volume through Catalog Explorer, then install them using a path like `/Volumes/main/engineering/libraries/mypackage-1.0.0-py3-none-any.whl`. The identity used for installation must have **READ VOLUME permission** on the specified volume.



Python **requirements.txt** files work with both workspace files and volumes in Databricks Runtime 15.0 and above. These files let you define multiple package dependencies in a single file, making it easier to maintain consistent environments across clusters. Upload the requirements.txt file and install it just like any other library—Azure Databricks automatically installs all listed packages.

For clusters with standard access mode, you must add library file paths to the `allowlist` before installation. This applies to both workspace files and volumes, ensuring admins approve the libraries used on shared compute.

Use init scripts for advanced configuration

Init scripts run shell commands during **cluster startup**, before the Spark driver and executors start. While Databricks **doesn't recommend** using init scripts for library installation—cluster-scoped

libraries provide a better approach—init scripts prove useful for system-level **configuration** that libraries can't handle.

You might use init scripts to install system packages with `apt-get`, configure environment variables, or set up monitoring agents. For example, an init script could install a specialized database driver that requires system libraries, then configure connection parameters through environment variables. The script runs every time the cluster starts, ensuring your configuration persists across restarts.

Store init scripts in Unity Catalog volumes for clusters running Databricks Runtime 13.3 LTS and above. Create a shell script file, upload it to a volume, then configure the cluster to run the script by specifying its path like `/Volumes/main/engineering/scripts/setup.sh`. For standard access mode, add the init script path to the `allowlist` before configuring the cluster.

Init scripts execute sequentially in the order you specify. If any script returns a non-zero exit code, the cluster fails to start. This failure protection prevents clusters from running with incomplete or incorrect configuration. You can troubleshoot failed init scripts by configuring cluster log delivery and examining the init script logs.

Consider init scripts as a last resort for configuration needs that cluster-scoped libraries and cluster policies can't address. Using cluster policies to set environment variables and Spark configurations often provides a simpler, more maintainable solution than init scripts.

Configure libraries for standard access mode

Clusters configured with **standard access mode** provide the strongest security and isolation in Azure Databricks. This mode requires explicit approval for libraries and init scripts to prevent unauthorized code execution on shared compute resources.

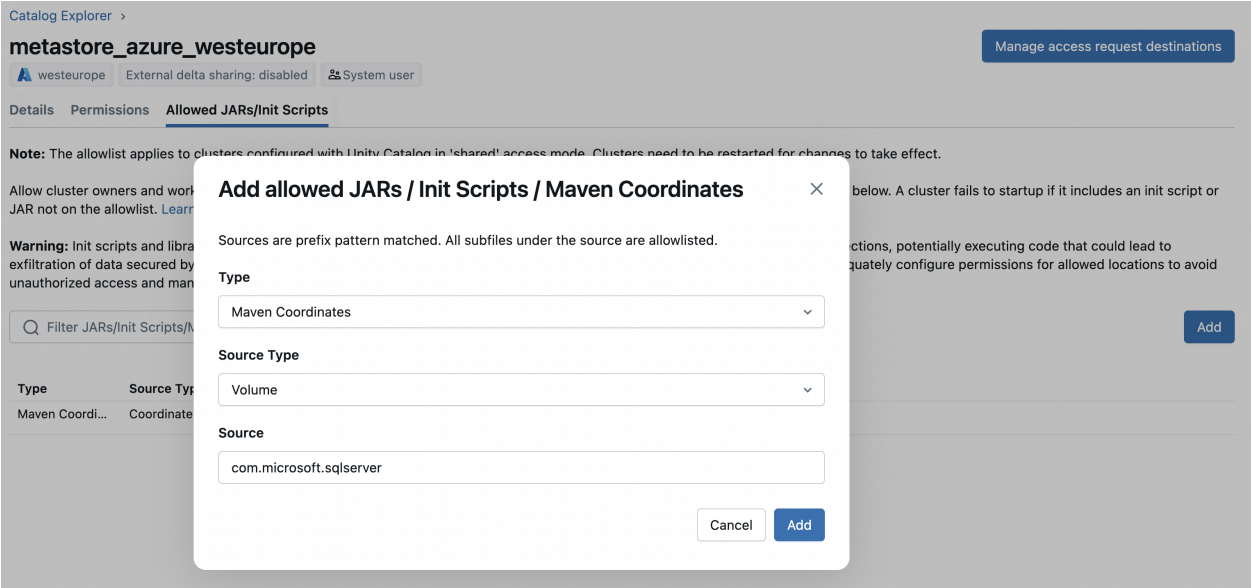
Before installing Maven libraries or JAR files on standard access mode clusters, a **metastore admin** must add them to the `allowlist`. Maven coordinates go on the `allowlist` using the format `groupId:artifactId:version`. You can `allowlist` all versions of a library with `groupId:artifactId`, or all artifacts in a group with just `groupId`. For JAR files stored in volumes or object storage, `allowlist` the file path or directory path.

Init scripts require separate `allowlist` entries even if stored in the same location as JAR files. When allow listing a path, Azure Databricks uses prefix matching—adding `/Volumes/prod-libraries/` to the `allowlist` permits all files and subdirectories within that location. Include a trailing slash to prevent unintended prefix matches at the directory level.

The `allowlist` only grants permission to use a path for library or init script installation. You still need appropriate data access permissions. For volumes, the installer identity must have `READ VOLUME` permission. For standard access mode, the cluster owner's identity validates these permissions during library installation.

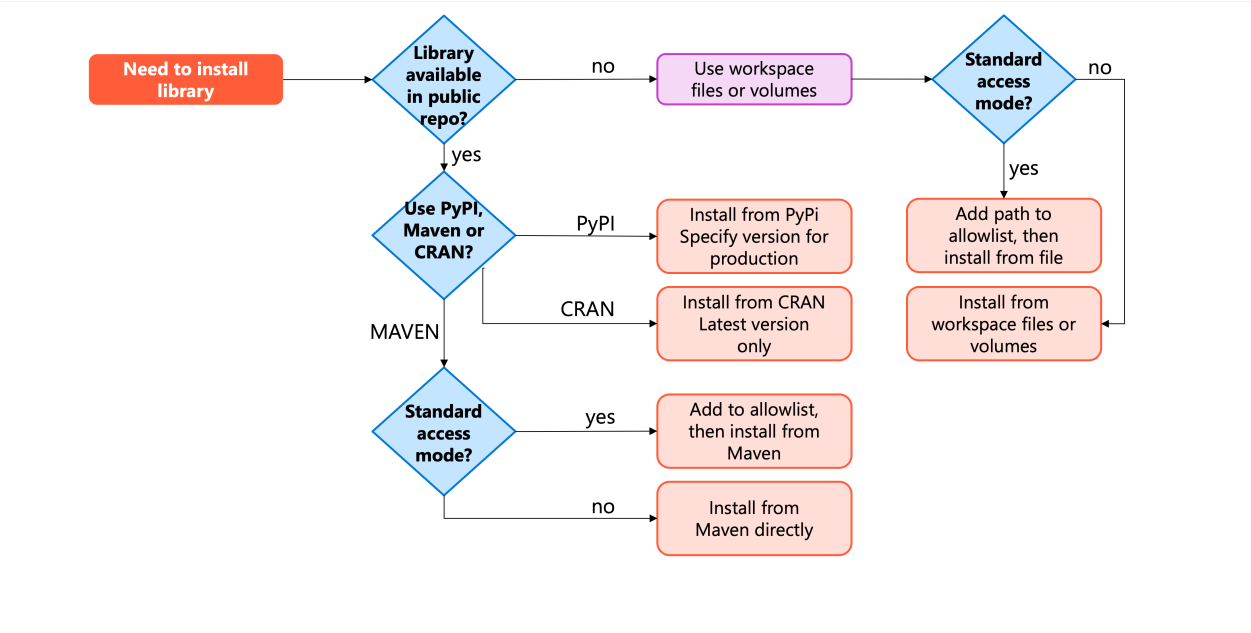
To configure the `allowlist`, metastore admins use Catalog Explorer, selecting the metastore settings and navigating to the **Allowed JARs/Init Scripts** section. This centralized control ensures

that security teams can review and approve all libraries used across the organization's compute resources, maintaining governance without blocking productivity.



Choose the right installation method

Different library installation methods suit different scenarios. The following diagram illustrates a decision flow to help you select the appropriate installation approach:



6. Configure compute access

Configure compute access

Completed

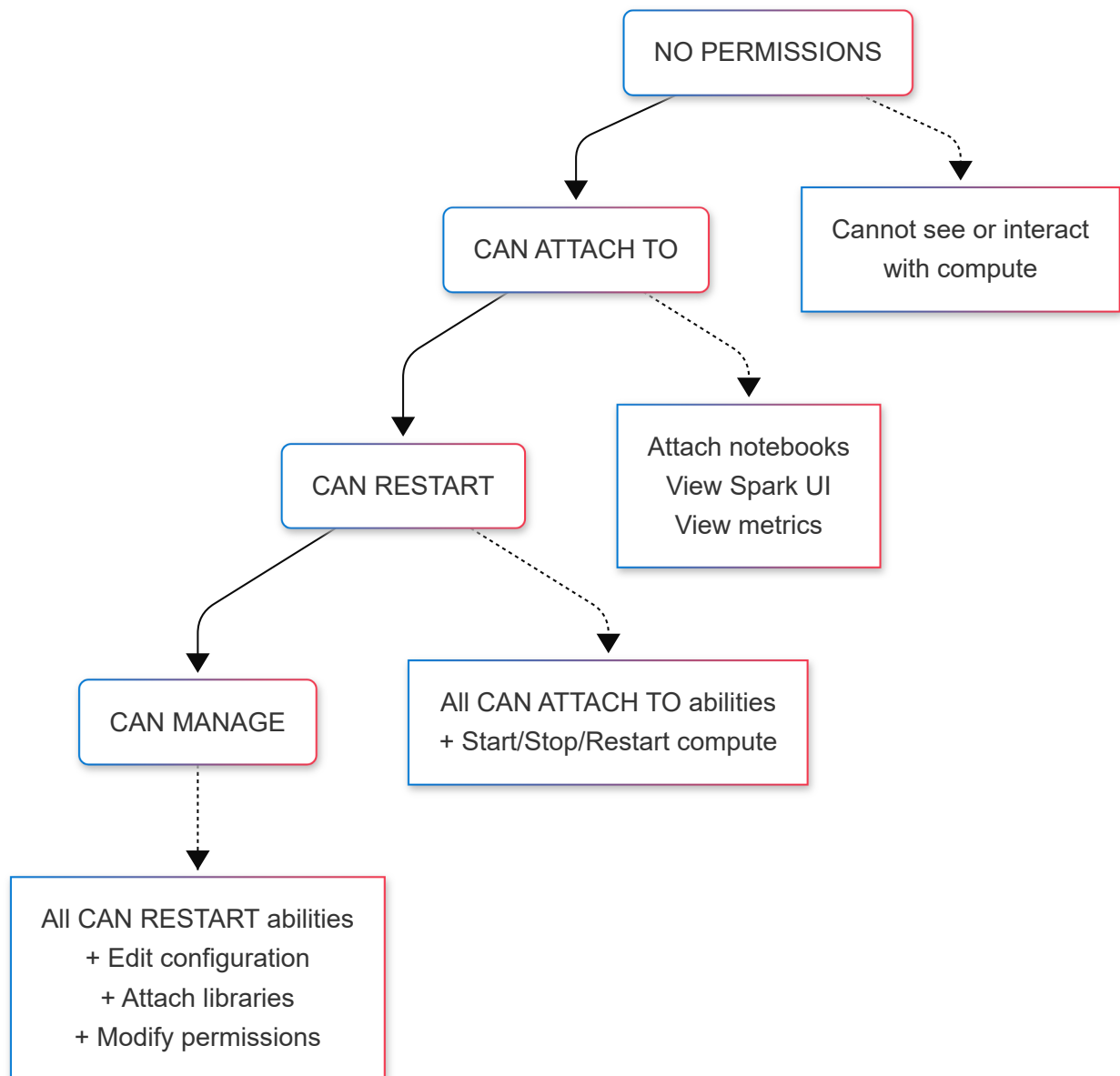
Managing access to compute resources protects sensitive data while enabling collaboration across your data engineering teams. You control who can attach notebooks, restart clusters, or modify compute configurations through **permission levels** assigned in the Azure Databricks workspace.

Proper access configuration prevents unauthorized users from viewing driver logs containing secrets, controls who can consume compute resources, and ensures teams work efficiently without conflicting infrastructure changes. With **Unity Catalog** enabled, you can assign compute resources to entire groups with automatic permission scoping.

Understand compute permission levels

Azure Databricks offers **four permission levels** for compute resources, each granting progressively more capabilities. These permissions operate within your workspace and are distinct from Azure subscription-level access controls.

The following diagram shows the hierarchy of compute permissions and the capabilities each level provides:



NO PERMISSIONS prevents any interaction with the compute resource. Users can't see the compute in their workspace, attach notebooks to it, or view metrics or logs. This default state ensures compute resources remain private until you explicitly grant access.

CAN ATTACH TO enables basic compute usage. Users with this permission can attach their notebooks to the compute, view the Spark UI to monitor job execution, and access compute metrics for performance analysis. However, they can't control the compute lifecycle—no starting, stopping, or restarting. This permission level works well for analysts who need to run queries but shouldn't manage infrastructure.

With this permission level, users on compute configured with **No isolation shared access mode** can view service account keys in log4j files. Because of this security consideration, No isolation shared access mode is legacy and should be avoided. **Standard** or **Dedicated** access modes provide better isolation.

CAN RESTART includes everything from **CAN ATTACH TO** plus lifecycle management capabilities. Users can terminate, start, and restart the compute resource. This permission suits team leads or senior engineers who need to manage compute availability without full administrative control. You might grant **CAN RESTART** to someone who needs to restart a misbehaving cluster but shouldn't change its configuration.

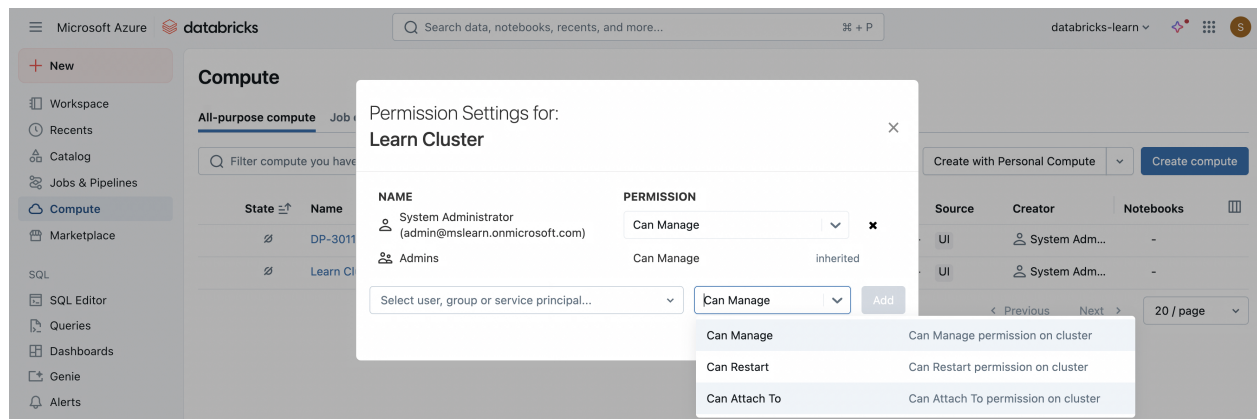
CAN MANAGE provides complete control over the compute resource. Beyond all **CAN RESTART** capabilities, users can edit compute configuration, attach libraries, resize the cluster, and modify permissions for other users. Workspace admins automatically receive **CAN MANAGE** permission on all compute resources. Users who create a compute resource become its owner with **CAN MANAGE** rights.

By default, only users with **CAN MANAGE** permission can view **driver logs** on job clusters, dedicated access mode clusters, and standard access mode clusters. This restriction exists because driver logs expose secrets through stdout and stderr streams. You can adjust this behavior using the Spark configuration property `spark.databricks.acl.needAdminPermissionToViewLogs`, though changing it may create security risks.

Configure permissions in the workspace

You set compute permissions through the Azure Databricks workspace UI, not the Azure portal. This workspace-level configuration operates independently from Azure subscription permissions, though users still need workspace access to benefit from compute permissions.

To configure permissions on a compute resource, select **Compute** from the workspace sidebar. Locate the compute resource you want to manage and select the **Permissions** option from its actions menu. The permissions dialog shows current access grants and allows you to add or modify permissions.



Select **Add** to grant new permissions. Search for **users** or **groups** by name or email address. Choose the appropriate permission level from the dropdown menu— **CAN ATTACH TO** , **CAN RESTART** , or **CAN MANAGE** . Multiple users and groups can have different permission levels on the same compute resource, giving you flexibility in access control.

When you modify permissions, changes take effect immediately. Users gain or lose access without requiring a compute restart. This immediate application means you can quickly respond to team changes, onboard new members, or revoke access when someone leaves a project.

Consider granting permissions to **groups** rather than individual users when possible. Group-based permissions simplify management as team membership changes. When someone joins the data engineering team, adding them to the appropriate group automatically grants the correct compute access. Similarly, removing someone from the group revokes their permissions without manual cleanup.

Manage workspace-level entitlements

Beyond individual compute permissions, Azure Databricks provides **workspace-level entitlements** that control what users can do across the entire workspace. These entitlements differ from compute permissions—they govern user capabilities rather than access to specific resources.

The **unrestricted cluster creation** entitlement allows non-admin users to create clusters without size restrictions. By default, only workspace administrators can create clusters of any size. When you enable this entitlement for a user or group, they gain the ability to provision clusters with any number of worker nodes and any VM configuration, but they still don't receive workspace admin privileges.

This entitlement follows the principle of least privilege by letting you grant cluster creation capabilities without elevating users to full workspace administrators. Users with this entitlement can create compute resources that match their workload requirements but can't perform other administrative functions like managing workspace settings, viewing all users' notebooks, or modifying account-level configurations.

To grant the unrestricted cluster creation entitlement:

1. In your workspace, select **Settings** from the sidebar.
2. Navigate to **Identity and access > Users or Groups**.
3. Locate the user or group you want to grant the entitlement to.
4. Select the user or group, then find the **Entitlements** section.
5. Enable the **Allow cluster creation** entitlement.

When designing your access control strategy, consider which users genuinely need unrestricted cluster creation. Data engineers building production pipelines often require this capability, while data analysts might only need access to pre-configured shared clusters. Grant this entitlement to roles and teams rather than individuals when possible to simplify management as your organization grows.

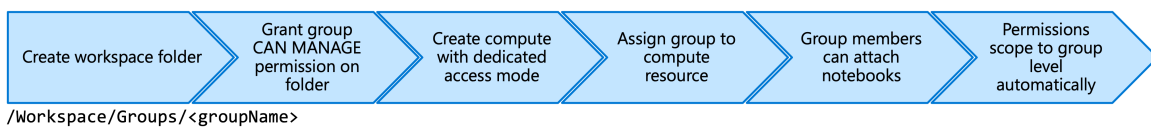
Dedicated group access mode

Standard access mode allows multiple users to share compute resources securely, with Lakeguard providing isolation between user workloads. **Dedicated access mode** takes a different approach by

assigning the entire compute resource to a single user or group, with user permissions automatically scoping down to match the assigned entity's permissions.

This **permission scoping** enables secure group collaboration on workloads that standard access mode doesn't support, including **Databricks Runtime for ML**, **RDD APIs**, and **R language** execution. When a user connects to a group cluster, their individual permissions temporarily reduce to only what the group possesses, preventing privilege escalation while maintaining necessary functionality.

The following diagram illustrates how to set up dedicated group access:



To create a compute resource dedicated to a group, you must enable Unity Catalog on your workspace and use Databricks Runtime 15.4 or above. The assigned group needs **CAN MANAGE** permission on a workspace folder where members can store notebooks, MLflow experiments, and other workspace artifacts used on the group cluster.

When creating the compute, expand the **Advanced** section and select **Dedicated (formerly: Single-user)** under **Access mode**. In the **Single user or group** field, choose the group that should own this resource. Only members of the selected group can attach notebooks to this compute resource.

System Administrator's Cluster

Policy

Unrestricted

☒ Multi node ☐ Single node

Access mode

Single user or group access

Dedicated (formerly: Single user)

System Administrator

Dedicated (formerly: Single user)

All languages

Standard (formerly: Shared)

Python, Scala, SQL

No isolation shared

All languages

For effective group cluster management, create a dedicated **workspace folder** at

/Workspace/Groups/<groupName> for each group using this pattern. Grant the group **CAN MANAGE**

permission on this folder. All notebooks, experiments, and workspace assets the group uses should reside in this folder to avoid permission errors.

Group clusters introduce specific behavior changes you should understand. When you create data objects using the group cluster, the group becomes the object's owner, not the individual user. For example, if you run `CREATE SCHEMA human_resources` on a group cluster, the group owns that schema. Individual user permissions aren't enforced because all group members access shared Spark APIs and the same compute environment.

Best practices for compute access

Apply the **principle of least privilege** when granting compute permissions. Start with the minimum permission level users need to accomplish their work. Grant `CAN ATTACH TO` for analysts running queries, `CAN RESTART` for engineers managing cluster availability, and reserve `CAN MANAGE` for administrators and resource owners.

Use **groups** instead of individual user assignments whenever possible. Create groups that mirror your organizational structure—data engineering teams, analytics teams, or project-specific groups. Grant permissions to these groups, then manage membership through the groups themselves. This approach scales better as your organization grows and reduces the likelihood of orphaned permissions when people change roles.

For production workloads, create **dedicated compute resources with restricted access**. Production clusters shouldn't be widely accessible to development users. Limit `CAN MANAGE` permissions to the specific team responsible for that workload, and grant `CAN ATTACH TO` only when users have legitimate business needs to run production queries.

Pay attention to **driver log access**, especially when debugging applications that use secrets. The default configuration restricts log viewing to users with `CAN MANAGE` permissions, protecting secrets from exposure. If you must grant broader log access by setting `spark.databricks.acl.needAdminPermissionToViewLogs` to `false`, ensure secrets are managed through Databricks secret scopes with appropriate access controls rather than being hardcoded.

When using **group clusters**, establish clear folder structures and permission patterns before deployment. Create the group workspace folders, assign appropriate permissions, and document the expected usage patterns. Ensure MLflow tracking URIs point to group folders and that AutoML runs specify group-accessible experiment directories. This upfront organization prevents permission errors and simplifies troubleshooting.

Monitor compute access through **audit logs** regularly. Review who has `CAN MANAGE` permissions on critical resources, verify that former team members no longer have access, and ensure permission grants align with current organizational needs. The `system.access.audit` table provides detailed access history for compliance and security reviews.

Avoid using No isolation shared access mode entirely. This legacy mode exposes service account keys to users with `CAN ATTACH TO` permissions and lacks the security guarantees of standard or

dedicated access modes. If you encounter existing No isolation shared clusters, migrate them to standard or dedicated access mode to improve security posture.

7. Exercise - Select and Configure Compute in Azure Databricks

<https://learn.microsoft.com/en-us/training/modules/select-and-configure-compute/7-exercise-select-configure-compute>

Exercise - Select and Configure Compute in Azure Databricks

Completed

Now it's your chance to select and configure compute in Azure Databricks. In this lab, you create and configure an all-purpose cluster in Azure Databricks, install libraries both cluster-scoped and notebook-scoped, and use the faker library to generate and analyze synthetic patient admission records with PySpark.

Note

To complete this lab, you need an [Azure subscription](#) in which you have administrative access.

Launch the exercise and follow the instructions.

[Launch Exercise](#)

8. Module assessment

<https://learn.microsoft.com/en-us/training/modules/select-and-configure-compute/8-knowledge-check>

Module assessment

Completed

9. Summary

<https://learn.microsoft.com/en-us/training/modules/select-and-configure-compute/9-summary>

Summary

Completed

Selecting and configuring compute resources effectively determines both the success and cost of your Azure Databricks workloads. Throughout this module, you explored the compute options available and learned how to match them to specific scenarios. Serverless compute provides the fastest path to productivity with minimal overhead, while classic compute offers complete control when you need specialized features. SQL warehouses optimize analytical queries, and job clusters ensure automated workflows run efficiently without idle costs.

Performance configuration extends beyond choosing a compute type. You discovered how node types, autoscaling settings, and termination policies balance cost with responsiveness. Memory-optimized instances handle large joins more efficiently, while compute-optimized instances excel at CPU-intensive transformations. Photon acceleration doubles query performance for SQL workloads, and instance pools reduce startup time when justified by usage patterns.

Access management and library installation complete the compute configuration picture. Permission levels enable the principle of least privilege, granting users exactly the access they need without unnecessary risk. Dedicated group access modes bring secure collaboration to workloads requiring RDD APIs or R language. Library installation from repositories, files, or volumes ensures your code has the dependencies it needs while maintaining security through allowlist controls.

As you implement these patterns in your organization, start with serverless options for new workloads, then move to classic compute only when specific limitations require it. Monitor actual usage to optimize configurations over time, apply permission controls thoughtfully, and maintain consistent library management practices across teams. The compute decisions you make today shape the performance, security, and cost efficiency of your data platform for months to come.