*Last updated on* ***Dec 9, 2025***

# Best practices for performance efficiency

This article covers best practices for **performance efficiency**, organized by architectural principles listed in the following sections.

## 1. Vertical scaling, horizontal scaling, and linear scalability

Before we get into the best practices, let's look at a few distributed computing concepts: horizontal scaling, vertical scaling, and linear scalability.

- **Vertical scaling**: Scale vertically by adding or removing resources from a single machine, typically CPUs, memory, or GPUs. This typically means stopping the workload, moving it to a larger machine, and restarting it. There are limits to vertical scaling: There may not be a bigger machine, or the price of the next bigger machine may be prohibitive.
- **Horizontal scaling**: Scale horizontally by adding or removing nodes from a distributed system. When the limits of vertical scaling are reached, the solution is to scale horizontally: Distributed computing uses systems with multiple machines (called clusters) to run workloads. It is important to understand that for this to be possible, the workloads must be prepared for parallel execution, as supported by the engines of the Databricks Data Intelligence Platform, Apache Spark, and Photon. This allows multiple inexpensive machines to be combined into a larger computing system. When more compute resources are needed, horizontal scaling adds more nodes to the cluster and removes them when they are no longer needed. While technically there is no limit (and the Spark engine does the complex part of load balancing), large numbers of nodes do increase management complexity.

- **Linear scalability**, meaning that when you add more resources to a system, the relationship between throughput and resources used is linear. This is only possible if the parallel tasks are independent. If not, intermediate results on one set of nodes will be needed on another set of nodes in the cluster for further computation. This data exchange between nodes involves transporting the results over the network from one set of nodes to another set of nodes, which takes considerable time. In general, distributed computing has some overhead for managing the distribution and exchange of data. As a result, small data set workloads that can be analyzed on a single node may be even slower when run on a distributed system. The Databricks Data Intelligence Platform provides flexible computing (single node and distributed) to meet the unique needs of your workloads.

## 2. Use serverless architectures

### Use serverless compute

With serverless compute on the Databricks Data Intelligence Platform, the compute layer runs in the customer's Databricks account. The services are fully managed and continuously enhanced by Databricks. In addition to customers only paying for what they use, this results in improved productivity:

- Cloud administrators no longer have to manage complex cloud environments, such as adjusting quotas, creating and maintaining network resources, and connecting to billing sources. They can focus their time on higher-value projects instead of managing low-level cloud components.
- Users benefit from near-zero cluster startup latency and improved query concurrency.

Databricks provides managed services for different workloads:

- Serverless SQL warehouses for SQL workloads

  Workspace admins can create serverless SQL warehouses that enable instant compute and are managed by Databricks. Use them with Databricks SQL queries just as you usually would with the original Databricks SQL warehouses. Serverless compute comes with a very fast startup time for SQL warehouses, and the infrastructure is managed and optimized by Databricks.

Ask Assistant

- Serverless jobs for efficient and reliable workflows

  Serverless compute for jobs allows you to run your Databricks job without configuring and deploying infrastructure. With serverless compute, you focus on implementing your data processing and analytics pipelines, and Databricks efficiently manages compute resources, including optimizing and scaling compute for your workloads. Autoscaling and Photon are automatically enabled for the compute resources running your job.

  You can monitor the cost of jobs that use serverless compute for jobs by querying the billable usage system table.

- Serverless compute for notebooks

  If your workspace is enabled for serverless interactive compute, all users in the workspace have access to serverless compute for notebooks. No additional permissions are required.

## Use an enterprise grade model serving service

Mosaic AI Model Serving provides a unified interface to deploy, govern, and query AI models. Each model you serve is available as a REST API that you can integrate into your web or client application.

Model serving provides a highly available and low-latency service for deploying models. The service automatically scales up or down to meet demand changes, saving infrastructure costs while optimizing latency performance. This functionality uses serverless compute.

# 3. Design workloads for performance

## Understand your data ingestion and access patterns

From a performance perspective, data access patterns - such as “aggregations versus point access” or “scan versus search” - behave differently depending on the data size. Large files are more efficient for scan queries, and smaller files are better for searches, because you need to read less data to find the specific row(s).

Ask Assistant

For ingestion patterns, it is common to use DML statements. DML statements are most performant when the data is clustered, and you can simply isolate the section of data. It is important to keep the data clustered and isolatable during ingestion: consider keeping a natural time sort order and apply as many filters as possible to the ingest target table. For append-only and overwrite ingestion workloads, there isn't much to consider because it's a relatively cheap operation.

The ingestion and access patterns often point to an obvious data layout and clustering. If not, decide what is more important to your business and focus on how to better achieve that goal.

## Use parallel computation where it is beneficial

Time to value is an important dimension when working with data. While many use cases can be easily implemented on a single machine (small data, few and simple computational steps), there are often use cases that need to process large data sets, have long run times due to complicated algorithms, or need to be repeated 100s and 1000s of times.

The cluster environment of the Databricks platform is a great environment for efficiently distributing these workloads. It automatically parallelizes SQL queries across all nodes of a cluster and it provides libraries for Python and Scala to do the same. Under the hood, the engines Apache Spark and Photon engines analyze the queries, determine the optimal way of parallel execution, and manage the distributed execution in a resilient way.

In the same way as batch tasks, Structured Streaming distributes streaming jobs across the cluster for best performance.

One of the easiest ways to use parallel computing is with Lakeflow Spark Declarative Pipelines. You declare a job's tasks and dependencies in SQL or Python, and then Lakeflow Spark Declarative Pipelines handles execution planning, efficient infrastructure setup, job execution, and monitoring.

For data scientists, pandas is a Python package that provides easy-to-use data structures and data analysis tools for the Python programming language. However, Pandas does not scale out to big data. Pandas API on Spark fills this gap by providing pandas equivalent APIs that work on Apache Spark.

In addition, the platform comes with parallelized machine learning algorithms in the standard machine learning library MLlib. It supports out-of-the-box multi-GPU usage. Deep learning can

Ask Assistant

also be parallelized using DeepSpeed Distributor or TorchDistributor.

## Analyze the whole chain of execution

Most pipelines or consumption patterns involve a chain of systems. For example, with BI tools the performance is impacted by several factors:

- The BI tool itself.
- The connector that connects the BI tool and the SQL engine.
- The SQL engine to which the BI tool sends the query.

For best-in-class performance, the entire chain must be considered and selected/tuned for best performance.

## Prefer larger clusters

Plan for larger clusters, especially if the workload scales linearly. In this case, using a large cluster for a workload is not more expensive than using a smaller cluster. It's just faster. The key is that you rent the cluster for the duration of the workload. So, if you spin up two worker clusters and it takes an hour, you are paying for those workers for the full hour. Similarly, if you spin up a four worker cluster and it only takes half an hour (this is where linear scalability comes in), the costs are the same. If costs are the primary driver with a very flexible SLA, an autoscaling cluster is usually the cheapest, but not necessarily the fastest.

**NOTE**

Preferring large clusters is not required for serverless compute because it automatically manages clusters.

## Use predictive optimization

To improve performance, tables require regular maintenance, such as optimizing the layout of the data, cleaning up old versions of data files that are no longer needed, and updating the clustering of the data. To ensure optimal performance, some of these tasks require a good understanding of data access patterns across the platform.

Ask Assistant

Databricks Unity Catalog governs all reads and writes for the tables it manages and is aware of all query patterns across the platform. Based on these patterns, predictive optimization can optimize tables according to how the data is actually being used, typically resulting in significant performance improvements. In addition, predictive optimization eliminates the need to manually manage maintenance operations for Delta tables on Databricks. The platform automatically identifies tables that would benefit from maintenance operations and runs them for the user.

Databricks recommends you enable predictive optimization for your account, catalog, or schema, if it isn't enabled already (as with managed tables).

## Use Unity Catalog managed tables

Tables in Unity Catalog can be created as managed tables or external tables. To create external tables you must specify the object storage location, and you are responsible for maintaining and optimizing these tables. For managed tables, Databricks manages the entire data lifecycle, including file layout and the automatically enabled predictive optimization, which typically results in significant performance improvements.

It is recommended to use Unity Catalog managed tables for all tabular data managed in Databricks.

## Use native Spark operations

User-defined functions (UDFs) are a great way to extend the functionality of Spark SQL. However, don't use Python or Scala UDFs if a native function exists:

- Spark SQL
- PySpark

Reasons:

- Serialization is required to transfer data between Python and Spark. This significantly slows down queries.
- Increased effort to implement and test functionality that already exists in the platform.

Ask Assistant

If native functions are missing and should be implemented as Python UDFs, use Pandas UDFs. Apache Arrow ensures data moves efficiently back and forth between Spark and Python.

## Use native platform engines

Photon is the engine on Databricks that provides fast query performance at low cost – from data ingestion, ETL, streaming, data science, and interactive queries – directly on your data lake. Photon is compatible with Apache Spark APIs, so getting started is as easy as turning it on – no code changes and no lock-in.

Photon is part of a high-performance runtime that runs your existing SQL and DataFrame API calls faster, reducing your total cost per workload. Photon is used by default in Databricks SQL warehouses.

## Understand your hardware and workload type

Not all cloud VMs are created equal. The various families of machines offered by cloud providers are all different enough to matter. There are obvious differences - RAM and cores - and more subtle differences - processor type and generation, network bandwidth guarantees, and local high-speed storage versus local disk versus remote disk. There are also differences in the "spot" markets. These should be understood before deciding on the best VM type for your workload.

> **NOTE**
> This is not required for serverless compute because serverless compute automatically manages clusters.

## Use caching

Caching stores frequently accessed data in a faster medium, reducing the time required to retrieve it compared to accessing the original data source. This results in lower latency and faster response times, which can significantly improve an application's overall performance and user experience. By minimizing the number of requests to the original data source, caching helps reduce network traffic and data transfer costs. This efficiency gain can be particularly beneficial for applications that rely on external APIs or pay-per-use databases. It can help

Ask Assistant

spread the load more evenly across the system, preventing bottlenecks and potential downtime.

There are several types of caching available in Databricks. Here are the characteristics of each type:

- **Use disk cache**

  The disk cache (formerly known as "Delta cache") stores copies of remote data on the local disks (for example, SSD) of the virtual machines. It can improve the performance for a wide range of queries but cannot be used to store the results of arbitrary subqueries. The disk cache automatically detects when data files are created or deleted and updates its contents accordingly. The recommended (and easiest) way to use disk caching is to choose a worker type with SSD volumes when configuring your cluster. Such workers are enabled and configured for disk caching.

- **Avoid Spark Caching**

  The Spark cache (by using `.persist()` and `.unpersist()`) can store the result of any subquery data and data stored in formats other than Parquet (such as CSV, JSON, and ORC). However, using incorrect locations in a query can consume all available memory and significantly slow down queries. As a rule of thumb, avoid Spark caching.

- **Query Result Cache**

  Per cluster caching of query results for all queries through SQL warehouses. To benefit from query result caching, focus on deterministic queries that for example, don't use predicates such as `= NOW()`. When a query is deterministic, and the underlying data is in Delta format and unchanged, SQL Warehouses will return the result directly from the query result cache.

- **Databricks SQL UI caching**

  Per user caching of all queries and legacy dashboard results in the Databricks SQL UI.

## Use compaction

Delta Lake on Databricks can improve the speed of reading queries from a table. One way is to coalesce small files into larger ones. You trigger compaction by running the OPTIMIZE command. See Optimize data file layout.

Delta Lake provides options for automatically configuring the target file size for writes and for OPTIMIZE operations. Databricks automatically tunes many of these settings, and enables features that automatically improve table performance by seeking to right-size files:

- **Auto compact** combines small files within Delta table partitions to automatically reduce small file problems. Auto compaction occurs after a write to a table has succeeded and runs synchronously on the cluster that has performed the write. Auto compaction only compacts files that haven't been compacted previously.
- **Optimized writes** improve file size as data is written and benefit subsequent reads on the table. Optimized writes are most effective for partitioned tables, as they reduce the number of small files written to each partition.

See Control data file size for more details.

## Use data skipping

Data skipping can significantly improve query performance by skipping over data that doesn't meet the query criteria. This reduces the amount of data that needs to be read and processed, leading to faster query execution times.

To achieve this, data skipping information is automatically collected when you write data to a Delta table (by default Delta Lake on Databricks collects statistics on the first 32 columns defined in your table schema). Delta Lake on Databricks uses this information (minimum and maximum values) at query time to provide faster queries. See Data skipping.

The following techniques can be applied for data skipping:

- Z-ordering, a technique for collocating related information in the same set of files. This co-locality is automatically used on Databricks by Delta Lake data-skipping algorithms. This behavior significantly reduces the amount of data Delta Lake must read.

- Liquid clustering simplifies data layout decisions and optimizes query performance. It will replace partitioning and z-ordering over time. Databricks recommends liquid clustering for all new Delta tables. Liquid clustering provides the flexibility to redefine clustering keys without rewriting existing data, allowing data layouts to evolve with analytical needs over time. Databricks recommends liquid clustering for all new Delta tables.

  Ask Assistant

  Tables with the following characteristics benefit from liquid clustering:

    - Filtered by columns with high cardinality.
    - With significantly skewed data distribution.
    - That grow rapidly and require maintenance and tuning effort.
    - With concurrent write requests.
    - With access patterns that change over time.
    - Where a typical partition key could leave the table with too many or too few partitions.

For more details and techniques see the Comprehensive Guide to Optimize Databricks, Spark, and Delta Lake Workloads.

## Avoid over-partitioning

In the past, partitioning was the most common way to skip data. However, partitioning is static and manifests itself as a filesystem hierarchy. There is no easy way to change partitions as access patterns change over time. Often, partitioning leads to over-partitioning - in other words, too many partitions with too small files, resulting in poor query performance.

Databricks recommends that you do not partition tables below 1TB in size, and that you only partition by a column if you expect the data in each partition to be at least 1GB.

In the meantime, a better choice than partitioning is Z-ordering or the newer Liquid Clustering (see above).

## Optimize join performance

- Consider **range join optimization**.

  A range join occurs when two relations are joined using a point in an interval or an interval overlap condition. The range join optimization support in Databricks Runtime can bring orders of magnitude improvement in query performance but requires careful manual tuning.

- Use **adaptive query execution**.

  Adaptive query execution (AQE) is query re-optimization that occurs during query execution. It has 4 major features:

Ask Assistant

  - Dynamically changes sort merge join into broadcast hash join.
  - Dynamically coalesces partitions after shuffle exchange.
  - Dynamically handles skew in sort merge join and shuffle hash join.
  - Dynamically detects and propagates empty relations.

  It is recommended to keep AQE enabled. Different features can be configured separately.

For more details, see the Comprehensive guide to optimize Databricks, Spark and Delta Lake workloads.

### Run analyze table to collect table statistics

The `ANALYZE TABLE` statement collects statistics about tables in a specified schema. These statistics are used by the query optimizer to generate an optimal query plan, selecting the correct join type, selecting the correct build side in a hash-join, or calibrating the join order in a multi-way join.

Predictive optimization automatically runs `ANALYZE` (Public Preview), a command for collecting statistics, on Unity Catalog managed tables. Databricks recommends enabling predictive optimization for all Unity Catalog managed tables to simplify data maintenance and reduce storage costs. See Predictive optimization for Unity Catalog managed tables.

## 4. Run performance testing in the scope of development

### Test on data representative of production data

Run performance testing on production data (read-only) or similar data. When using similar data, characteristics like volume, file layout, and data skew should be similar to the production data, since this has a significant impact on performance.

### Consider prewarming resources

Regardless of the query and data format, the first query on a cluster is always slower than subsequent queries. This is because all the different subsystems are starting up and reading all

Ask Assistant

the data they need. Prewarming has a significant impact on performance test results:

- **Prewarm clusters**: Cluster resources need to be initialized on multiple layers. It is possible to prewarm clusters: Databricks pools are a set of idle, ready-to-use instances. When cluster nodes are created using these idle instances, cluster startup and autoscaling times are reduced.
- **Prewarm caches**: When caching is part of the setup, the first run ensures that the data is in the cache, speeding up subsequent jobs. Caches can be prewarmed by running specific queries to initialize caches (for example, after a cluster restart). This can significantly improve the performance of the first few queries.

So, to understand the behavior for the different scenarios, test the performance of the first execution with and without prewarming and of subsequent executions.

### Identify bottlenecks

Bottlenecks are areas in your workload that could degrade the overall performance as the load increases in production. Identifying these at design time and testing against higher workloads will help to keep the workloads stable in production.

## 5. Monitor performance

### Monitor query performance

Monitoring query performance helps you understand how resources are being used by different queries. You can identify queries that are running slowly, allowing you to pinpoint performance bottlenecks in your system. You can also identify queries that are consuming significant system resources, potentially leading to instability or downtime. This information helps you optimize resource allocation, reduce waste, and ensure that resources are being used efficiently.

The Databricks Data Intelligence Platform has various monitoring capabilities (see Operational Excellence - Set up monitoring, alerting and logging), some of which can be used for performance monitoring:

- **Query Profile**: Use the query profile feature to troubleshoot performance bottlenecks during query execution. It provides visualization of each query task and related metrics

Ask Assistant

such as time spent, number of rows processed, and memory used.

- **SQL Warehouse Monitoring**: Monitor SQL warehouses by viewing live statistics, peak query count charts, running clusters charts, and query history table

## Monitor streaming workloads

Streaming monitoring enables you to analyze data and detect issues as they occur, providing real-time insights into your system's performance and behavior. By analyzing streaming data, you can identify trends, patterns, and opportunities for optimization. This can help you fine-tune your system, improve resource utilization, and reduce costs.

For streaming queries, use the built-in Structured Streaming monitoring in the Spark UI or push metrics to external services using the Apache Spark Streaming Query Listener interface.

## Monitor job performance

Job monitoring helps you identify and address issues in your Lakeflow Jobs, such as failures, delays, or performance bottlenecks. Job monitoring provides insights into job performance, enabling you to optimize resource utilization, reduce wastage, and improve overall efficiency.

Ask Assistant