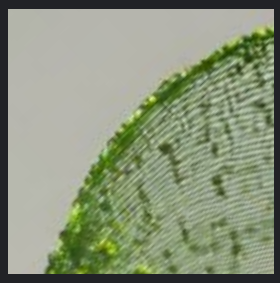
Google Cloud



A practitioner's guide to Apache Spark®

in the agentic era

Mastering the borderless lakehouse, data science, and real-time pipelines.

Introduction

Apache Spark® in the agentic era

The ubiquity of Apache Spark

Apache Spark has been the foundation of large-scale data processing for over a decade.

Its open-source ecosystem has grown to over 2,000 global contributors and more than 80% of Fortune 500 companies now rely on it to power their most critical data workloads.

2,000 global contributors

80% of Fortune 500 companies rely on Spark to power their most critical data workloads

From massive batch ETL pipelines and real-time streaming ingestion to interactive analytics and distributed machine learning, Spark's in-memory processing framework has made it the industry's default programmatic engine.

Behind this ubiquity stands the data engineer, whose critical labor in building, maintaining, and optimizing pipelines and Spark jobs makes this consumption possible. Yet as we enter the agentic era, these pipelines are being put to the test.

The shift from human scale to agent scale

Enterprise data architectures are undergoing a fundamental shift. For the past two decades, data platforms served human-scale consumption: business analysts writing SQL queries, data scientists training models, and executives viewing static dashboards. In this human-centric model, a highly productive data team might execute a few dozen queries per hour.

In the agentic era, this model breaks. Autonomous AI agents—designed to perceive, reason, and act on behalf of the enterprise—operate at a non-linear, exponential scale. A single agentic workflow can trigger thousands of concurrent, multi-hop queries per second to ground its reasoning, validate schemas, and execute transactions.

This shift from human scale to agent scale introduces severe physical and economic bottlenecks. If a query takes an extra second to execute or costs a fraction of a cent more, latency and compute costs balloon exponentially when multiplied across a fleet of autonomous agents. To survive the physics of agent scale, the underlying data platform must be optimized from the ground up.

System of Action vs. System of Intelligence

Legacy data architectures were designed as Systems of Intelligence—passive repositories built to analyze historical data and generate reports. They severed the link between operational systems (the "now") and analytical systems (the "history"), forcing organizations to rely on stale, high-latency batch updates.

The agentic era demands a System of Action built on an elastic, Zero-Idle-Compute architecture. Because autonomous agents operate on event-driven, sporadic, and highly concurrent patterns, traditional "always-on" streaming clusters result in massive resource underutilization and cost leakage. Real-time streaming is no longer a niche requirement, but is more critical than ever: an agent can't wait for a nightly batch process run to resolve an out-of-stock event or detect a fraudulent transaction. It must accurately and precisely process real-time streams, reason over historical context, and execute transactional writes in a single, closed loop.

The role of Apache Spark® in the agentic era

As the world transitions to AI-powered data systems, Apache Spark is becoming increasingly important. Standardizing on Managed Service for Apache Spark on Google Cloud delivers three core capabilities required for agentic scale.

1. High-fidelity multimodal ingestion

AI agents require conformed, high-fidelity data to act without hallucinating. Spark is the foundational engine that transforms raw, unstructured, and multimodal data (such as images, documents, and audio) into clean, structured, and vector-embedded tables.

2. The primary runtime for AI-generated code

The rise of automated code generation—where LLMs write code on behalf of developers—has positioned Spark as the primary target runtime for AI-generated data pipelines. LLMs natively generate PySpark code because of Spark's clean, standardized, and highly expressive APIs. To support this automated code-generation loop, the underlying Spark runtime must be exceptionally reliable, secure, and performant, able to execute AI-generated code without manual human tuning. Managed Service for Apache Spark provides this exact zero-ops execution environment, serving as the programmatic bedrock of Google's Agentic Data Cloud.

3. A foundation of portability and openness

In an ever-evolving landscape of cloud providers and models, organizations value choice. By standardizing on open-source APIs like Apache Spark and open-table formats like Apache Iceberg®, Google Cloud ensures that enterprise data pipelines remain cloud-agnostic and highly portable. As AI models, LLMs, and autonomous agent frameworks evolve at a breakneck pace, this open data foundation ensures organizations can seamlessly swap, upgrade, or integrate new AI technologies without having to rebuild their underlying data pipelines or pay a proprietary "walled garden" tax.

Your architectural roadmap

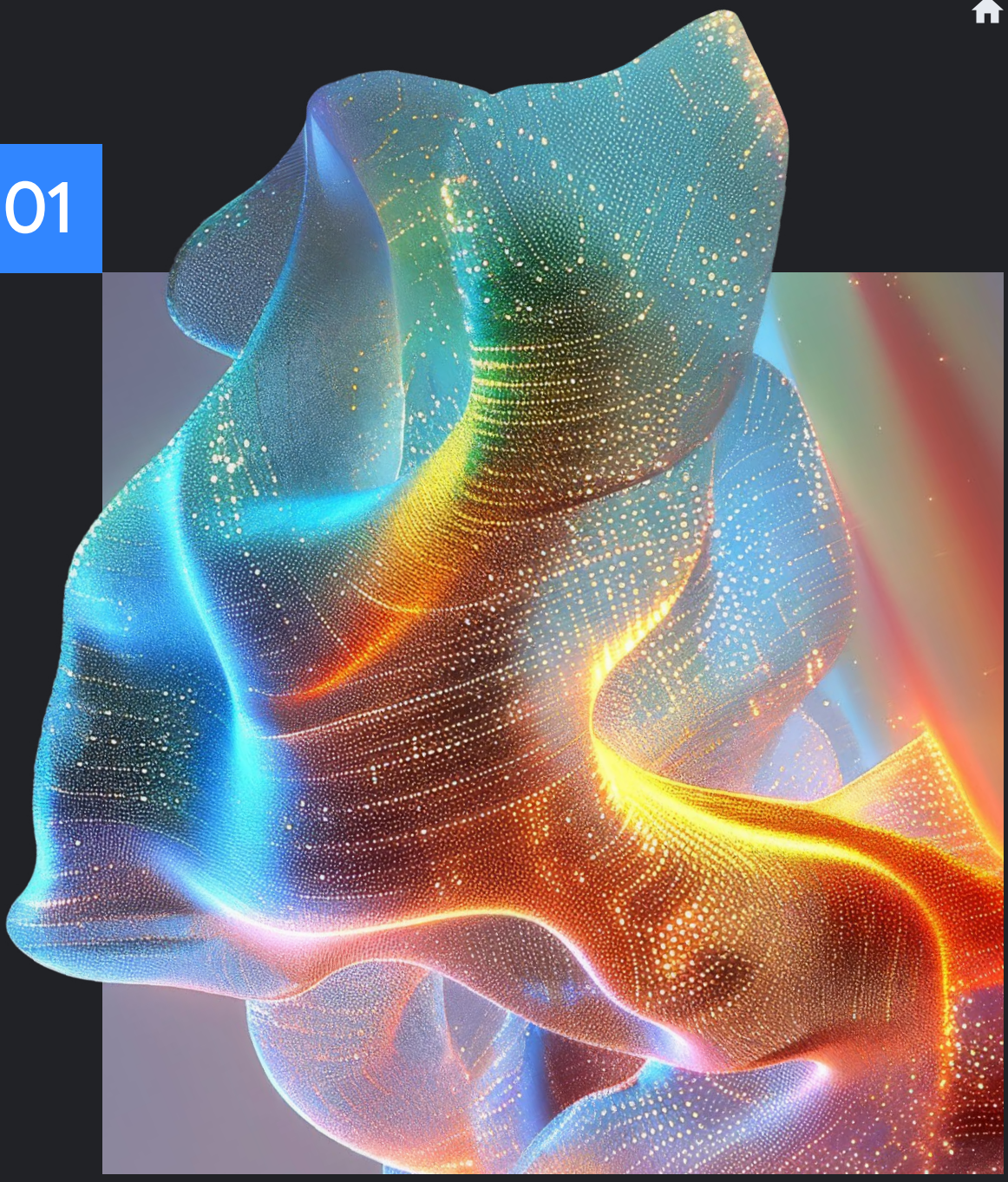
This guide is written specifically for data engineers, data scientists, and data architects who are transitioning enterprise data platforms to support agentic workloads.

It explores how Managed Service for Apache Spark has been engineered to eliminate operational overhead and performance bottlenecks, and provides the concrete, low-level architecture underpinning Google Cloud's data processing capabilities.

Packed with structured, step-by-step architectural workflows and direct links to the Lakehouse Solutions GitHub Repository and interactive Codelabs, it's your technical guide to operating Managed Service for Apache Spark in the agentic era.

Table of contents

01



A better way to run Apache Spark®

How Google Cloud makes Apache Spark easier, smarter, and faster.

1.1

Easier

Deployment flexibility and a unified workspace

To make Apache Spark® easier, Google Cloud eliminates the operational burden of maintenance and optimization—allowing developers to focus entirely on their code.

Choice of serverless and clusters

Data practitioners can run Spark in two distinct ways on Google Cloud, depending on their operational requirements.

Serverless (Spark Jobs as a Service) provides true zero-ops execution where the underlying infrastructure is completely managed. Developers submit their Spark code and the platform automatically provisions, scales, and tears down resources, billing only for the exact seconds of job runtime. This is highly suited for ephemeral, "spiky," or batch-oriented agentic workloads. The serverless form factor is also ideal for interactive Spark in notebooks, referred to as "serverless Spark interactive sessions."

Conversely, managed clusters (Spark Clusters as a Service) provide fine-grained control over the underlying virtual machines, custom initialization actions, and specific network topologies. This mode is critical for long-running, continuous-stream processing where predictable resource allocation and dedicated compute capacity are required. Customers also use it for running a pipeline (DAG of jobs) where the cluster is created once and multiple jobs are run on it.

Reimagining the practitioner experience

With its investments in Gemini and agentic experiences, Google Cloud is uniquely positioned to redefine the developer experience on Spark, making it agentic, seamless, and intuitive.

Boilerplate coding, data cleaning, and manual glue-code writing are delegated to autonomous agents, allowing developers to focus on innovation. Developers can start and finish their entire workflow within a single, unified workspace.

Whether you prefer Colab Enterprise, JupyterLab instances in Gemini Enterprise Agent Platform Workbench, or your local Visual Studio Code environment, Managed Service for Apache Spark meets you where you work, allowing you to initiate interactive serverless Spark sessions from your IDE of choice. The serverless Spark runtime provides near-instant, resource contention-free, dedicated, scalable Spark compute for rapid experimentation. Colab and VS Code also offer native code generation and troubleshooting capabilities, significantly speeding up delivery.

Freedom of choice and the Data Agent Kit

We believe developers should be free to choose whatever toolchain works for them. Managed Service for Apache Spark supports your preferred IDE (VS Code, JupyterLab, Colab Enterprise) and LLM (Gemini, Claude, Codex). To enable this, Google Cloud has released open-source Spark skills and Model Context Protocol (MCP) support as part of the Data Agent Kit. The Data Agent Kit allows data practitioners to connect their favorite IDEs directly to remote Spark interactive sessions, monitor execution plans, and author, test, and schedule pipelines from a single pane of glass.

1.2

Smarter

Autonomous resource optimization and cost controls

Google Cloud uses smart automation to eliminate the operational burden of managing, scaling, and fine-tuning Apache Spark®. This autonomous optimization removes the costly "tuning tax" while actively preventing budget overruns.

Zero-scale clusters (master-only state)

In standard Apache Spark architectures, a cluster requires a master node that serves as central resource coordinator and secondary nodes (workers) to process data. Managed Service for Apache Spark introduces zero-scale clusters—when no data processing jobs are active, all worker nodes are automatically shut down and only the master node remains online to listen for new jobs.

When a new Spark job is submitted, the control plane instantly provisions new secondary workers to handle the heavy lifting. This guarantees zero idle cost for the secondary workers, so you no longer pay for expensive compute workers during lunch breaks, overnight gaps, or waiting periods between data deliveries.

Lifecycle management policies

To prevent accidental budget overruns when developers leave clusters running, the platform enforces automatic safety nets. Developers can set a strict lifespan when creating a cluster via time-to-live (TTL) policies (e.g., "delete this cluster exactly two hours after creation"). The cluster will self-destruct even if a developer forgets to turn it off.

Additionally, idle deletion policies monitor cluster activity. If no Spark jobs are detected for a specified window (e.g., 30 minutes of absolute silence), the cluster automatically deletes itself. Scheduled operations can also automatically spin clusters down at 6:00 PM every evening and bring them back up at 8:00 AM on weekdays.

Customers can even decide to stop the cluster (save the cluster state on GCS) and relinquish the VMs. Once the cluster is re-started, the VMs are recreated from the saved state. This eliminates the need to reinstall all the additional libraries, which in turn accelerates startup times.

Improved obtainability and autoscaling

To prevent job failures in managed Spark clusters due to stockouts of a single VM type, FlexVMs allow developers to specify a prioritized list of VM types for master, primary, and secondary worker nodes.

- **Autozone placement** simplifies infrastructure deployment by automatically selecting the optimal compute engine zone within a specified region based on real-time hardware availability.
- **Intelligent autoscaling** dynamically adjusts cluster hardware size in real time based on active queue backlogs and CPU utilization, choosing between performance-optimization or cost-optimization.

When using Managed Service for Apache Spark serverless, in addition to intelligent autoscaling, autotuning automatically optimizes a batch job's Spark configurations (such as shuffle partitions and executor configs) based on its historical trends, with built-in safety checks to prevent regressions.

1.3

Faster

Lightning Engine and vectorized execution

To make Apache Spark® faster, Google Cloud introduced Lightning Engine, a ground-up re-engineering of the Spark execution layer designed to maximize the efficiency of modern cloud hardware.

It delivers up to 4.9x faster performance than open-source Spark, and up to 2x the price-performance of the leading high-speed Spark alternative. This is achieved with zero code changes, only the addition of feature-specific configuration.

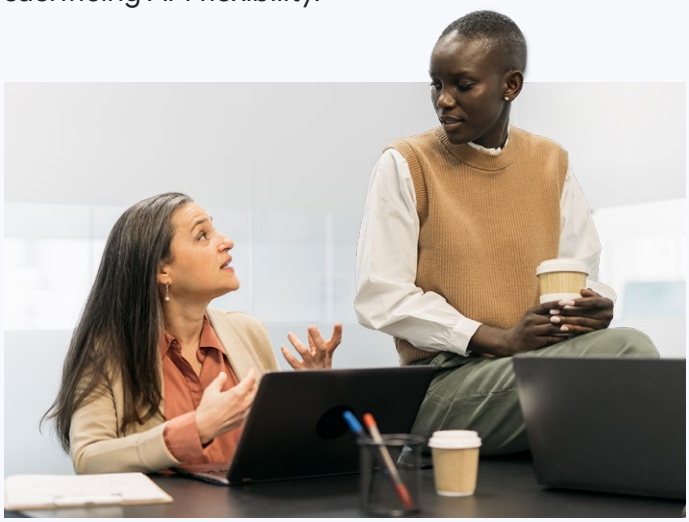
Traditional Spark executes physical plans within the Java Virtual Machine (JVM), which introduces heavy overhead from object creation, garbage collection, and virtual method dispatch. Lightning Engine bypasses the JVM for physical execution by compiling Spark physical plans directly into native C++ instructions using Apache Gluten and Velox. It leverages the Single Instruction, Multiple Data (SIMD) function on modern CPU architectures to process data in vectorized columns rather than row-by-row, maximizing hardware utilization and lowering total cost of ownership (TCO). Lightning Engine's performance enhancements go beyond native query execution.

Data shuffling is often the most expensive phase of a Spark query, creating an execution bottleneck with heavy CPU serialization and disk I/O. Lightning Engine incorporates an enhanced columnar shuffle manager that optimizes data serialization and network transfer, significantly reducing CPU overhead during wide transformations. To minimize redundant data retrieval from object storage, Lightning Engine also features built-in intelligent caching mechanisms that automatically cache hot data (such as Parquet footers, Apache Iceberg® metadata, and frequently accessed columns).

Lightning Engine supports most common enterprise workloads and environments out of the box, including:

- **Parquet, ORC, Iceberg, and Delta Lake table formats**
- **DataFrame, Dataset, SparkSQL, PySpark, and Scala APIs**
- **Debian, Ubuntu, Spark 3.5.7, and x86 architectures**

This broad compatibility ensures that organizations can run their workloads on Lightning Engine without refactoring existing codebase logic or sacrificing API flexibility.



Optimized storage connectors and I/O

Lightning Engine replaces standard Apache Hadoop® filesystem connectors with highly optimized, native connectors that target the primary bottlenecks of cloud object storage.

The re-engineered execution layer helps to reduce scan times and lower cloud costs in three key ways:

- **Direct path connection**
  It establishes a direct, bi-directional gRPC stream with Google Cloud Storage (GCS), skipping the Cloud Frontend (CFE) and allowing executors to perform seek operations and parallelized vectored reads (readV APIs) without reopening streams.
- **BigQuery direct Apache Arrow® consumption**
  Standard Apache Spark® connectors retrieve BigQuery data in Apache Arrow format and convert it into Spark's internal UnsafeRow format, incurring a heavy CPU conversion tax. Lightning Engine natively consumes Arrow format directly within its C++ execution engine.
- **Lexicographic listing**
  In open-source Spark, listing directories with millions of files requires recursive metadata calls, often resulting in millions of GCS API operations. Lightning Engine uses lexicographic listing APIs to fetch all file metadata in a few high-throughput calls at driver level, then distributes this metadata to executors via an optimized protocol.

Get hands on

Validate Lightning Engine performance

Experience the speed of native vectorized execution and autotuning firsthand. Run the step-by-step Lightning Engine codelab to test performance optimizations and observe execution metrics with zero code changes.

Run codelab

Advanced query optimizer rules

Lightning Engine incorporates an advanced, cost-based query optimizer that leverages custom rules to accelerate execution.

These advanced rules enable:

- **Partial aggregation pushdown**
  Pushes partial aggregations below the shuffle phase to help avoid expensive shuffle operations.
- **Optimized broadcast hash joins**
  Caches broadcast hash tables at the executor level instead of rebuilding them for every task, reducing the memory footprint.
- **History-based optimizations**
  Bypasses static plan limitations by automatically analyzing metadata from completed runs of the same or similar queries. The optimizer applies these historical metrics to adaptively tune physical execution strategies—such as join reordering and dynamic partition pruning—reducing slot time and query latency without requiring manual developer configuration.

TCO savings of serverless Apache Spark®

These architectural advantages translate into massive, proven TCO benefits, validated by an ESG economic benefits study in March 2025.

Apache Spark

18–60%

savings vs. alternative cloud-based managed clusters

21–55%

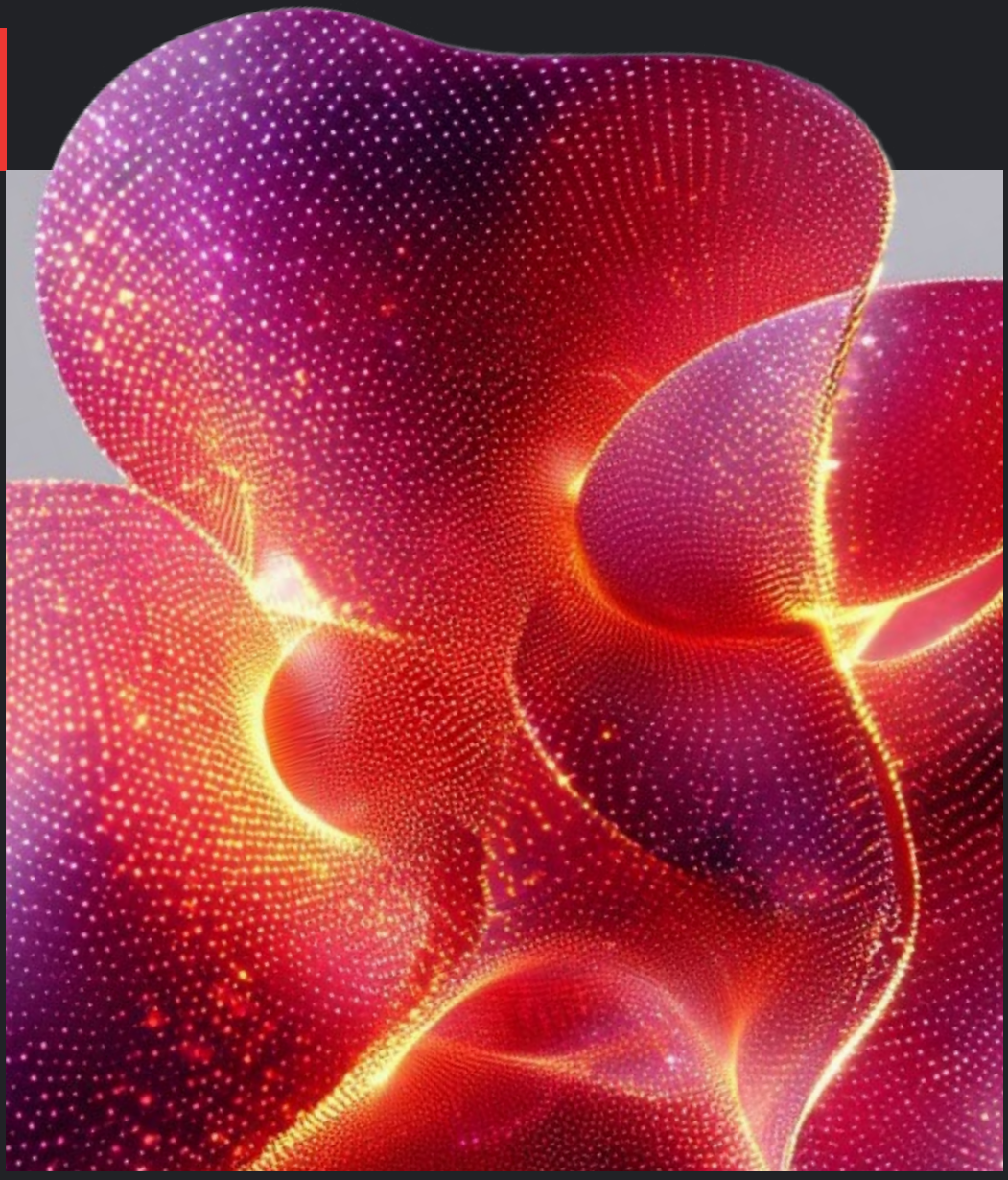
savings vs. alternative serverless Spark offerings

79–90%

savings when combining Managed Spark with BigQuery

Source: The Economic Benefits of Google Cloud Dataproc and Serverless Spark Versus Alternative Solutions, ESG, March 2025

02



Apache Spark® in the borderless lakehouse

The architecture underpinning Apache Spark's data processing capabilities.

Managed, interoperable storage with Apache Iceberg®

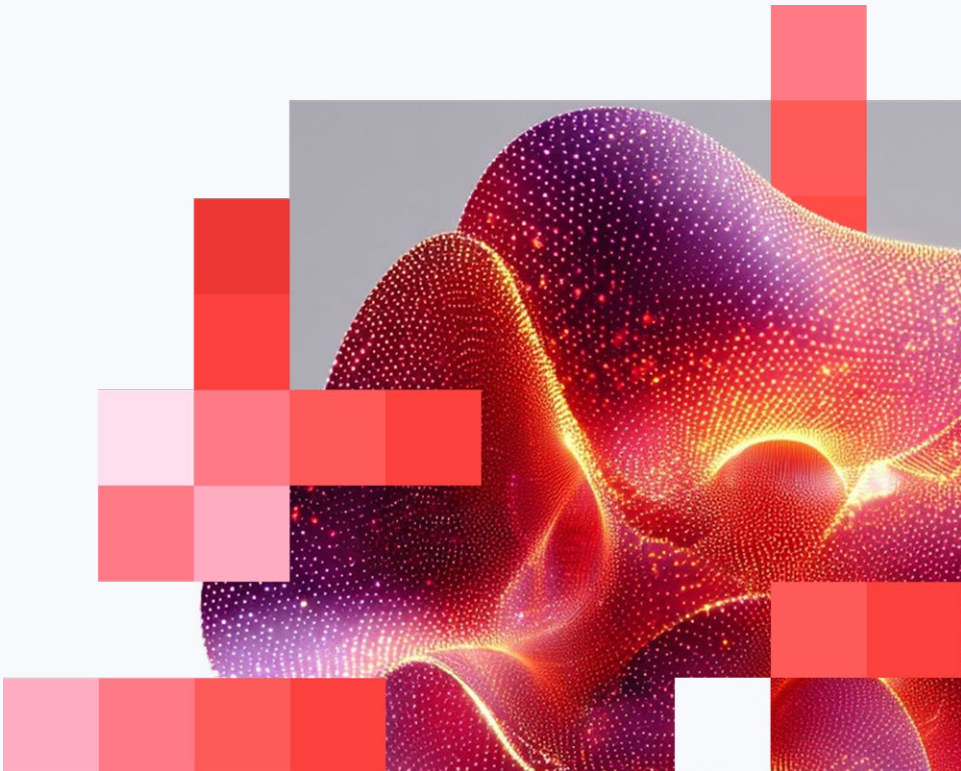
Apache Iceberg has emerged as the de facto open standard table format for the lakehouse, providing SQL-like ACID transactions, schema evolution, partition pruning, and time-travel capabilities on top of object storage. To govern and coordinate access to these tables across multiple compute engines, Google Cloud provides the Lakehouse runtime catalog—a highly available, scalable, serverless, standalone implementation of the Iceberg REST Catalog specification.

The Lakehouse runtime catalog acts as the single source of truth for table metadata, enabling seamless read/write interoperability between Managed Service for Apache Spark, BigQuery, and other open-source engines (e.g., Flink or Trino). Key capabilities of this unified catalog include:

- **Managed Iceberg lifecycle**
  Automated garbage collection and manifest maintenance to prevent metadata bloat and clean up orphaned data files.
- **Lakehouse performance boosts**
  Fast merge for Iceberg and Delta tables, optimized vacuum with advanced GCS integration, and faster compaction using partition-aware compaction and binary merging of Parquet files.
- **Unified access control**
  Completely eliminates security friction by delivering consistent, table-level ACLs via credential vending alongside fine-grained access control that spans Apache Spark® and BigQuery natively.

To delegate storage access without relying on proprietary, engine-specific agents, the Lakehouse runtime catalog uses a secure credential vending mechanism.

When a Managed Service for Apache Spark job queries an Iceberg table, the catalog does not grant the Spark executors broad read access to the underlying GCS bucket. Instead, the Lakehouse runtime catalog—integrated with the Knowledge Catalog, Google Cloud's unified data governance and metadata management service—evaluates the user's identity and provides the executors with short-lived, down-scoped IAM credentials, restricted to the specific catalog, namespace, and table ACLs required for that query. This implementation of least privilege access at the storage boundary prevents unauthorized data access even if a user attempts to bypass Spark APIs using raw file reads.



The medallion architecture on Google Cloud

The medallion architecture is the industry-standard design pattern for organizing data within a lakehouse, establishing a logical flow that refines raw data into high-value, conformed assets. On Google Cloud, this architecture is implemented using a combination of GCS, Lakehouse runtime catalog for Apache Iceberg, and BigQuery, processed via Managed Service for Apache Spark.

The bronze zone (raw fidelity)

Raw data lands in GCS. This is the immutable record from source systems.

The silver zone (enterprise truth)

Apache Spark® cleans and conforms the data, applying business rules. Crucially, we register the data as Apache Iceberg® tables in the Lakehouse runtime catalog.

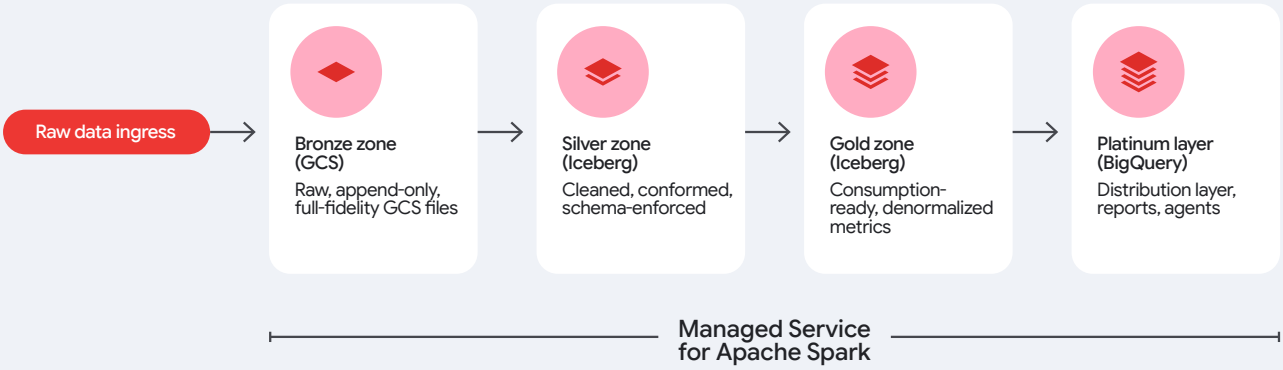
The gold zone (consumption ready)

Contains highly optimized, denormalized, and aggregated datasets stored in high-performance formats like Apache Iceberg, structured for maximum query efficiency.

The platinum layer (distribution and BI)

Serves as the semantic and distribution layer. Because the data is stored in Iceberg and registered in the Lakehouse runtime catalog, BigQuery can query the gold zone data directly. You achieve the massive concurrency of BigQuery SQL and Looker dashboards without moving, copying, or translating the data.

From raw to conformed: The medallion architecture data flow



Reference architecture: Unstructured data to Lakehouse for Apache Iceberg

AI agents require access to unstructured and multimodal data (images, PDFs, audio, etc.) to ground their reasoning. However, legacy architectures require this data to be copied into proprietary databases, creating data silos and security risks. Google Cloud delivers a unified pipeline for unstructured data processing that enables a zero-copy, multi-engine execution model.

How pipeline unification delivers multi-engine execution

1. Unstructured data ingestion
2. Managed Service for Apache Spark programmatic processing
3. Zero-copy, multi-engine execution

First, raw unstructured files land in GCS and are represented as GCS object tables or BigQuery ObjectRef types, preserving full-fidelity history.

Second, Managed Service for Apache Spark reads these object references, performs heavy-lifting ETL (such as OCR, document chunking, and metadata extraction), calls Gemini Enterprise Agent Platform APIs to generate vector embeddings, and writes the conformed features directly into Lakehouse Iceberg tables stored in GCS.

Third, because the data is stored in open Iceberg tables registered in the Lakehouse runtime catalog, different engines can act on the same data natively without moving, copying, or translating a single byte. BigQuery queries the Iceberg tables natively for high-performance SQL analytics, while Gemini Enterprise Agent Platform performs vector search and grounds autonomous agents in real-time, business-specific context directly from the Iceberg tables.

Get hands on

Build your borderless Lakehouse

Access the complete, runnable PySpark notebooks and Terraform scripts to automate the provisioning of your network, GCS buckets, and Lakehouse runtime catalog. Clone the templates directly from the **Lakehouse Solutions GitHub Repository** to deploy the Medallion architecture.

Visit repository

Get hands on

Deploy the Lakehouse runtime catalog lab

Access the complete, runnable PySpark notebooks and lab manuals to experience the Lakehouse runtime catalog. Clone the templates directly from the **Lakehouse Solutions GitHub Repository**.

Visit repository

Borderless Apache Iceberg® lakehouse interoperability

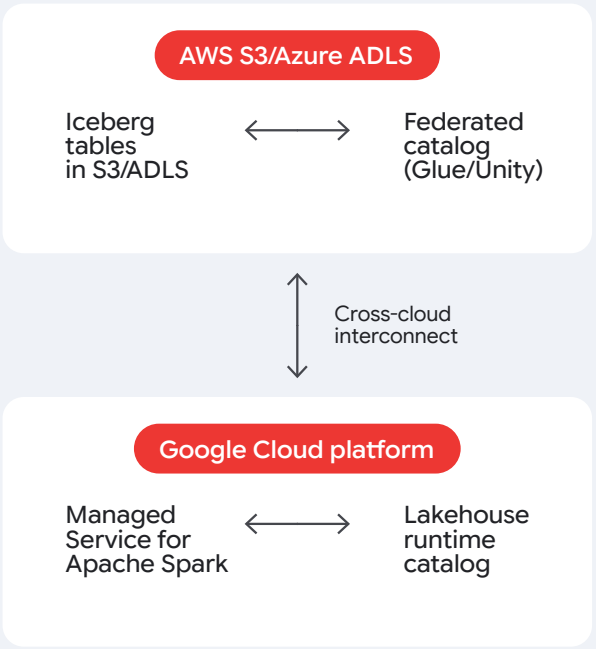
Enterprise data is frequently distributed across multiple cloud providers, creating fragmented data silos and forcing organizations into costly, high-latency data replication pipelines. Standardizing on Lakehouse for Apache Iceberg and the Lakehouse runtime catalog allows organizations to build a secure, high-performance cross-cloud lakehouse that enables multi-engine data access without the cost and latency of bulk data migration.

Through this cross-cloud architecture, the Managed Service for Apache Spark engine running on Google Cloud can access and process data stored in Amazon S3 or Azure Data Lake Storage (ADLS) natively with zero data movement. This is achieved through catalog federation and by establishing a dedicated, low-latency physical connection via cross-cloud interconnects, which significantly reduces data egress costs and latency between cloud boundaries.

When a Managed Service for Apache Spark job queries an Iceberg table stored in an external cloud bucket, the engine federates metadata queries using the Lakehouse runtime catalog. The catalog resolves the table schema and file locations, allowing the Apache Spark® executors to perform parallelized, vectored reads directly from the remote storage.

To optimize performance and eliminate redundant remote cloud egress fees on subsequent queries, the Spark executors cache the specific data blocks read (such as Parquet column chunks) in local SSD storage. Subsequent queries on the same dataset read directly from this high-speed local cache, accelerating query performance to near-native speeds. While local SSD caching incurs standard local storage costs, the architecture delivers a massive net reduction in TCO by eliminating recurring cross-cloud egress fees.

The cross-cloud lakehouse architecture



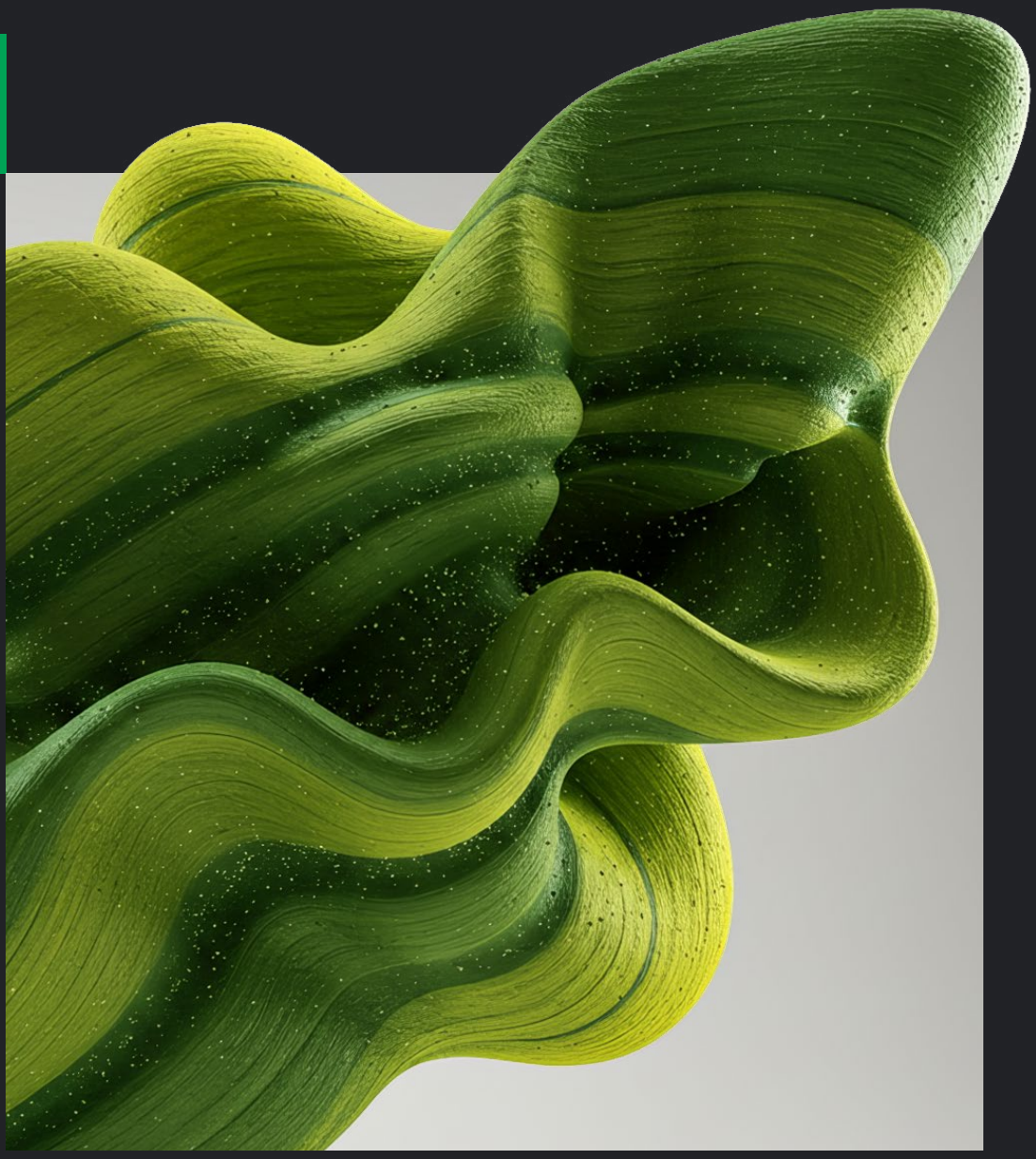
Get hands on

Deploy a secure, high-performance multi-cloud architecture

Experience cross-cloud data federation and processing firsthand. Run the step-by-step **Cross-cloud lakehouse codelab**.

Run codelab

03



Practical use cases and blueprints

A hands-on guide to the four leading use cases for Managed Service for Apache Spark.

3.1

Use case 1

Batch ETL at scale

Why Managed Service for Apache Spark is better

Processing massive, scheduled data volumes reliably requires an engine that can scale infinitely without manual infrastructure management. Here's how Managed Service for Apache Spark delivers.

Industry-leading price-performance

Delivers up to 4.9x faster performance than open-source Apache Spark® and up to 2x the price-performance of the previous market leader. These efficiency gains are achieved with zero code changes, requiring only the addition of feature-specific configuration.

Zero-ops administration

Eliminates cluster setup, capacity planning, and manual infrastructure management. Built-in autotuning dynamically optimizes internal software configurations—such as shuffle partitions and memory allocations—based on historical run metrics, eliminating the manual tuning tax and preventing out-of-memory errors.

Limitless cloud scaling

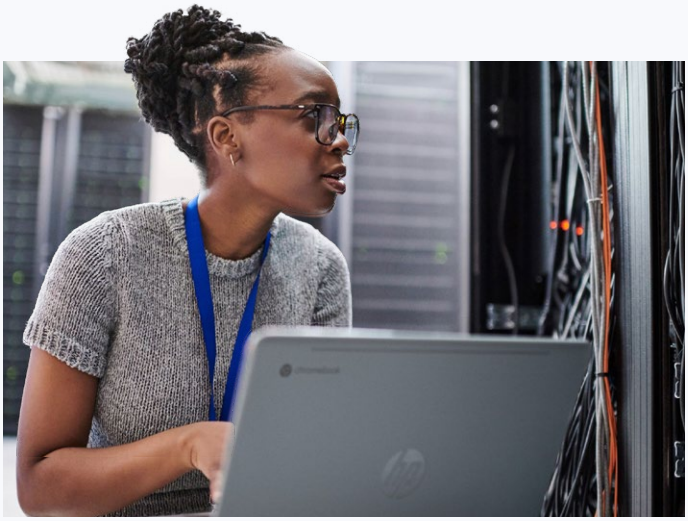
Serverless Spark automatically scales resources up and down, paying only for the exact seconds the job runs.

Apache Airflow® integration

Decouples pipeline orchestration from raw execution using Managed Service for Apache Airflow. Offloading tasks directly to the serverless Spark runtime prevents coordinator resource contention, ensuring highly resilient and isolated scheduling.

Performance optimizations

Lightning Engine, when enabled, can significantly improve the performance of batch ETL jobs without code changes. In addition, heavy batch ETL transformations can benefit significantly from GPU acceleration. By launching Spark with the RAPIDS Accelerator, data platform teams can run existing Spark batch applications on GPUs with zero code changes, accelerating physical execution plans natively.



Canonical ingestion pipelines for data mesh domains

In a decentralized data mesh architecture, individual business domains operate as independent units, owning their own data pipelines and products. To prevent fragmented architectures and operational bottlenecks, domains require standard, self-service infrastructure blueprints.

By using Managed Service for Apache Spark, organizations can establish canonical ingestion pipelines. These templated, reusable patterns allow domains to instantly read raw data from landing zones, apply standardized schema validation and cleansing, and write conformed tables directly to a shared Lakehouse for Apache Iceberg. This standardized approach slashes new domain onboarding time, enabling secure, self-service analytics to scale across petabytes of data without sacrificing enterprise governance.

Batch ETL blueprint

To implement the automated data mesh ingestion pattern, data platform teams can deploy automated pipelines orchestrated by Managed Service for Apache Airflow (Cloud Composer). Using the Airflow operator for Managed Service for Apache Spark in your Airflow DAG, Airflow triggers the serverless Spark batch and waits for the success/fail signal. This decouples your orchestration layer from your heavy compute layer, making your pipelines infinitely more resilient and scalable.

When the pipeline is submitted, Managed Service for Apache Spark control plane automatically applies history-based autotuning configurations. By adding the “--cohort” and “--autotuning-scenarios=auto” configurations, the engine dynamically optimizes shuffle partitions and memory configurations based on previous runs, eliminating the manual iterative tuning rigor that typically leads to OOM errors or disk spills.

Get hands on

Deploy the serverless batch pipeline

Access the complete, runnable PySpark notebooks and lab manuals to experience Managed Spark Serverless batches. Clone the templates directly from the **Lakehouse Solutions GitHub Repository.**

Visit repository

Get hands on

Deploy the orchestrated pipeline

Access the complete, runnable Airflow DAGs and lab manuals to experience pipeline orchestration. Clone the templates directly from the **Lakehouse Solutions GitHub Repository.**

Visit repository

3.2

Use case 2

Interactive analytics and data prep

Why Managed Service for Apache Spark is better

The iterative process of cleaning, exploring, and joining data often forces data professionals to constantly switch contexts between SQL editors and Python environments, disrupting analytical flow and introducing operational friction. Managed Service for Apache Spark eliminates this switching bottleneck by unifying SQL and PySpark within a single, secure notebook environment.

Unified SQL and Apache Spark®

Data scientists and analysts can use Colab Enterprise notebooks to seamlessly switch between BigQuery SQL for simpler queries and Spark SQL for more advanced transformations, all within an integrated development environment.

Custom internal platforms

For large-scale organizations, Spark can be integrated directly into custom internal platforms (such as experimentation dashboards) to run PySpark workloads on demand.

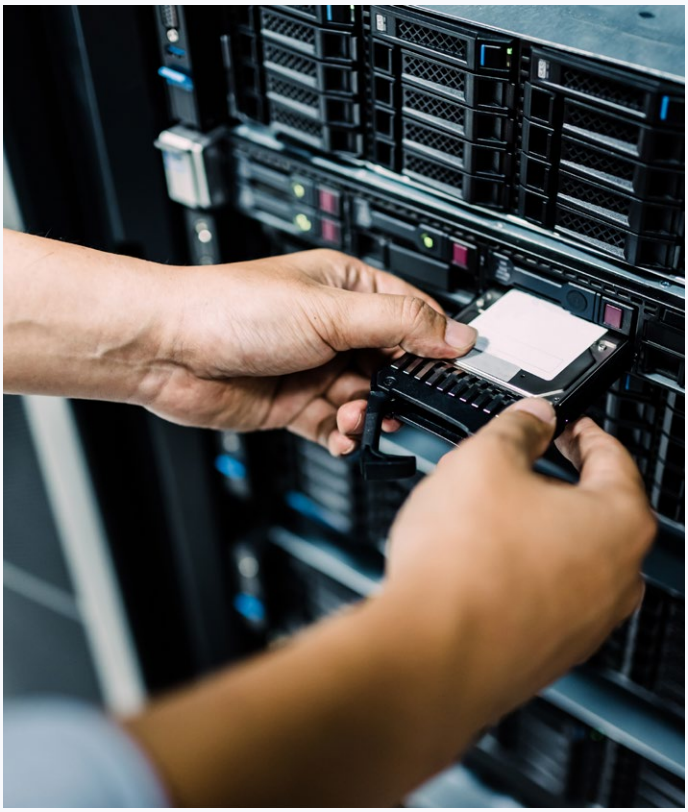
Ephemeral interactive sessions

Interactive serverless Spark sessions spin up in seconds, providing on-demand, scalable compute for rapid experimentation, then automatically tear down when idle to eliminate cluster costs.

Custom experimentation platform (AB console)

In large-scale digital enterprises, evaluating product changes requires running thousands of parallel A/B tests. Rather than product managers and data scientists having to write raw Apache Spark® scripts, platform teams can build a custom internal experimentation dashboard (an “AB console”) that abstracts raw data as clear outputs.

Under the hood, when a user configures an experiment or filters metrics across variables, the AB console pushes these interactive requests to Managed Service for Apache Spark, which provisions temporary compute resources, executes heavy-lifting PySpark workloads to calculate thousands of distinct metrics on demand, and returns low-latency insights directly to the UI. This democratizes data access while maintaining strict cost controls through automated idle deletion.



Interactive analytics blueprint

To implement interactive data preparation and feature engineering within a unified notebook environment, data practitioners can leverage Colab Enterprise notebooks. This allows them to seamlessly switch between BigQuery SQL for simpler queries and Spark SQL for more advanced transformations, all within an integrated development environment.

The serverless nature of Spark in BigQuery eliminates the need to manage and pay for permanent Spark clusters, addressing previous inefficiencies and reducing low-level configuration tasks.

Get hands on

Deploy the interactive analytics lab

Access the complete, runnable PySpark notebooks and lab manuals to experience interactive data preparation. Clone the templates directly from the **Lakehouse Solutions GitHub Repository**.

Visit repository

Get hands on

Deploy the Lakehouse runtime catalog lab

Access the complete, runnable PySpark notebooks and lab manuals to experience the Lakehouse runtime catalog for Apache Iceberg. Clone the templates directly from the **Lakehouse Solutions GitHub Repository**.

Visit repository

3.3

Use case 3

Data science and machine learning

Why Managed Service for Apache Spark is better

Accelerating the journey from feature engineering to model deployment requires a platform that eliminates the complexity of managing GPU drivers, CUDA versions, and machine learning (ML) libraries.

Pre-packaged ML runtimes

Specialized managed Apache Spark® image versions (Ubuntu-based, starting from 2.3) ship pre-packaged with NVIDIA GPU drivers (CUDA, cuDNN, NCCL) and common ML libraries (PyTorch, XGBoost, tokenizers, transformers), cutting cluster provisioning and setup time.

Dynamic Workload Scheduling (DWS)

Secures high-demand GPU resources (such as L4 and A100 families) reliably for SLA-sensitive AI/ML workloads, pausing workloads in a queue until the required capacity becomes accessible to prevent idle compute charges.

NVIDIA GPU acceleration

Run existing Spark applications on GPUs with zero code changes by launching Spark with the RAPIDS Accelerator, significantly accelerating physical execution plans natively.

Unified MLOps

Integrates Spark ML workflows directly with the Gemini Enterprise Agent Platform for consistent feature management, experiment tracking, and model registration.

Distributed hyperparameter tuning with Gemini Enterprise Agent Platform

Data scientists training complex deep learning models (such as recommendation algorithms) must evaluate thousands of hyperparameter combinations. Instead of performing these training runs sequentially on a single machine, platform teams can orchestrate distributed hyperparameter tuning jobs using Managed Service for Apache Spark and Gemini Enterprise Agent Platform.

Apache Spark® distributes the trial configurations across a dynamic pool of GPU-accelerated serverless workers, executing parallel training runs and logging metrics back to the Gemini Enterprise Agent Platform TensorBoard. This pattern significantly accelerates model convergence while maintaining strict cost controls through automated resource teardown.

Data science blueprint

For distributed feature engineering and training with Spark, data scientists and ML engineers can leverage interactive notebooks integrated with the Managed Service for Apache Spark ML runtime for rapid experimentation. Once validated, these machine learning workflows can be operationalized as Kubeflow pipelines on the Gemini Enterprise Agent Platform, with resulting models registered securely in the model registry. Downstream batch prediction workloads can then be executed efficiently on Managed Service for Apache Spark batches, with scheduling and workflow dependency management orchestrated natively by Managed Service for Apache Airflow.

For qualified ML workloads, GPU acceleration can be configured. By routing the execution through the RAPIDS Accelerator, the physical plan offloads heavy DataFrame sorting and hashing transformations directly to the executor's GPU cores. This configuration significantly accelerates feature preparation times, allowing data science teams to feed conformed arrays directly into deep learning models at scale.

Get hands on

Deploy and train ML models on Spark

Experience distributed ML and model training firsthand. Run the step-by-step **Spark ML models codelab**.

Run codelab

3.4

Use case 4

Structured streaming and near real-time processing

Why Managed Service for Apache Spark is better

Ingesting and processing high-velocity, real-time data streams requires an elastic infrastructure that can handle unpredictable spikes in data velocity without dropping data or requiring manual cluster resizing.

Predictable resource allocation

Google Cloud recommends launching continuous, long-running streaming workloads on managed clusters. This ensures predictable resource allocation, dedicated compute capacity, and lower baseline costs over extended execution windows.

Unified memory management

Lightning Engine's unified memory manager fluidly transfers memory between off-heap (used for the native C++ vectorized execution) and on-heap (Java) environments, ensuring custom UDFs run seamlessly alongside vectorized execution for maximum streaming throughput.

Optimized stream processing

Lightning Engine optimizations apply directly to structured streaming workloads.

Native messaging integration

Connects natively to messaging services like Managed Service for Apache Kafka without requiring complex network peering or the management of underlying brokers.

Near real-time data quality validation

In high-throughput retail and inventory systems, maintaining real-time accuracy of product metadata, pricing, and availability across thousands of active stores is critical. Rather than relying on slow, daily batch snapshots, platform teams can convert their batch ingestion pipelines into Spark Structured Streaming applications.

Using Managed Service for Apache Spark on managed clusters, the streaming application ingests incremental product updates from an Apache Kafka® topic. To ensure data integrity before writing to the lakehouse, the Apache Spark® application executes real-time, stateful data quality checks on the incoming streams against an external, low-latency price-availability API. This reduces processing time compared to traditional batch processing, ensuring that downstream analytical and operational systems operate on conformed, validated data.

Structured streaming blueprint

To initialize a high-throughput streaming execution, the Spark Structured Streaming application defines its source configurations to read continuously from the target messaging queue. The pipeline uses structured checkpoints stored in GCS, enabling transactional consistency and state recovery in the event of worker preemptions.

This execution is deployed on dedicated compute, ensuring that the processing loop maintains low-latency execution and predictable resource reservation. The application uses structured write streams to output conformed data directly to Apache Iceberg® tables, which are immediately registered in the Lakehouse runtime catalog for downstream consumption.

Get hands on

Deploy the streaming pipeline

Experience real-time data ingestion and processing firsthand. Run the step-by-step **Spark Structured Streaming codelab**.

Run codelab

04

Building enterprise operational readiness

Managing the developer experience, security, observability, and governance.

Operating Apache Spark® at enterprise scale requires robust, multi-layered security controls, comprehensive observability, business continuity, and unified governance. Managed Service for Apache Spark integrates natively with Google Cloud's enterprise framework to deliver these capabilities.

1

Developer experience

With the Data Agent Kit extension for VS Code, in addition to coding and troubleshooting assistance, you can author Spark code, notebooks and pipelines using just prompts.

To bring the power of generative AI directly into the operational lifecycle, Managed Service for Apache Spark integrates natively with Gemini Cloud Assist. When a complex Spark job fails or experiences performance degradation, Gemini Cloud Assist checks driver logs, executor metrics, and system telemetry to instantly identify potential root causes. It provides actionable, step-by-step recommendations to resolve failed or slow batch jobs, eliminating manual log analysis and troubleshooting.

2

Security and perimeter protection

VPC Service Controls (VPC-SC)

These establish a secure perimeter around your Spark resources, preventing unauthorized data exfiltration.

Customer-managed encryption keys (CMEK)

These allow organizations to encrypt data stored in GCS buckets, BigQuery tables, and temporary shuffle disks using KMS keys.

Secure VPC-only execution

Spark drivers and executors run without public IP addresses, communicating entirely over the private Google network. Inter-node Spark RPC communication is automatically encrypted and Spark authentication via shared secrets is enabled by default.

Non-root container execution

Drivers and executors run as non-root users inside secure Docker containers, eliminating the risk of privilege escalation.

Lakehouse runtime catalog security

This offers end user credentials as well as credential vending for authentication and catalog, namespace, and table ACLs, helping implement least privilege access.

3

Observability (logging, monitoring, alerting)

Cloud Logging and Cloud Monitoring

Driver and executor logs, along with system-level metrics (CPU, memory, disk, network), are streamed in real time to cloud logging and cloud monitoring.

Apache Spark® UI

A managed, serverless Spark UI is automatically spun up and available for troubleshooting live Spark jobs and interactive sessions.

Persistent History Server (PHS)

An optional, managed PHS can be provisioned and backed by a GCS log bucket. Spark clusters, serverless Spark batches, and interactive sessions on Managed Service for Apache Spark can be configured to write their event logs to this bucket, allowing developers to access the full Spark UI for deep-dive troubleshooting long after the compute resources have been torn down.

4

High availability and business continuity

Dynamic allocation of drivers and executors

When submitting a serverless batch, the resource manager dynamically allocates drivers and executors across multiple zones within the target region. If a zone experiences a hardware failure, the resource manager automatically provisions replacement executors in surviving zones, migrating active shuffle blocks asynchronously to ensure business continuity without job failure. For business continuity, there are various options and considerations based on RTO and RPOs.

Lakehouse runtime catalog

This features built-in high availability, automatic multi-region replication, and a managed backup and restore service, ensuring that your metadata plane is as resilient as your storage layer.

5

Governance and semantic context with Knowledge Catalog

Entries

As Spark jobs complete and create Apache Iceberg® tables in the Lakehouse runtime catalog, entries are automatically created in the Knowledge Catalog, and any updates are automatically synchronized.

Lineage

As Spark jobs execute transformations across the medallion layers, with lineage configuration in place (enabled via the lineage API and Spark session configurations), the platform automatically captures and records data lineage, tracing the flow of data from raw source files to final consumption tables.

Data profiling and quality

This offers the ability to profile conformed Iceberg tables, define data quality rules, configure and run quality jobs, monitor for quality thresholds and schema drift, and trigger alerts when quality thresholds are breached.

Semantic layer

This maps raw physical columns to conformed business definitions, ensuring that autonomous agents query data with full business understanding.

Conclusion

Modernizing your Apache Spark® workloads

Transitioning to Managed Service for Apache Spark allows data engineering and data science teams to abstract the underlying infrastructure and focus on pipeline logic and model development, with agent coding and troubleshooting assistance.

By leveraging serverless execution for dynamic scaling, Lightning Engine for vectorized C++ performance, and open-table formats for engine independence, organizations can build the resilient, high-performance data foundation required for the agentic era.

Across batch ETL, interactive data science, and real-time streaming, performance is the ultimate driver of developer velocity and cloud costs. Managed Service for Apache Spark delivers performance without the tuning tax across your entire data lifecycle.

Contributors

Matt Ryan
Product Marketing Manager, Google Cloud

Anagha Khanolkar
Solution Architect, Google Cloud

Suda Srinivasan
Group Product Manager, Google Cloud

Bhooshan Mogal
Senior Product Manager, Google Cloud

Brad Miro
Developer Relations, Google Cloud

Ajay Kumar
Global Practice Architect, Google Cloud

Next steps

Further learning in the agentic era

The path to agentic transformation starts with practical implementation.

Try Managed Service for Apache Spark for free

Get started with $300 in free credits to run your first job.

Try now

Deploy the blueprints on GitHub

Clone the Lakehouse Solutions GitHub Repository to access the Terraform scripts and template notebooks designed to run within Free Tier limits.

Visit GitHub

Contact us for expert advice

Get in touch