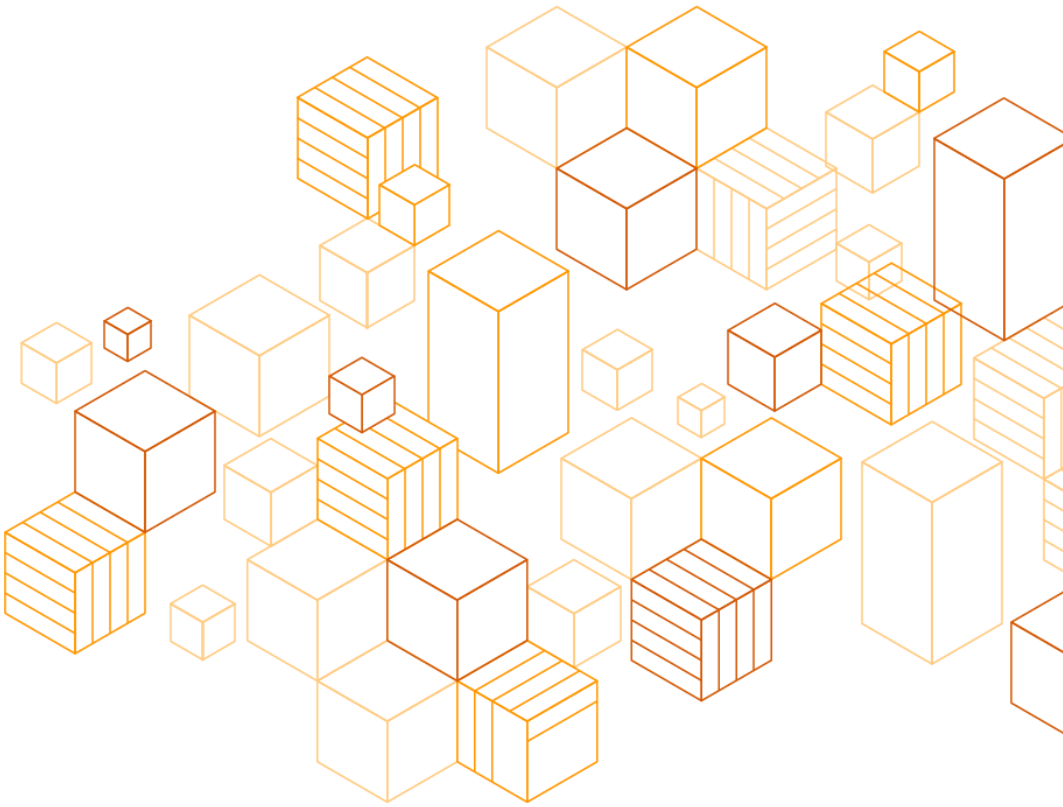
Amazon EMR Migration Guide

How to Move Apache Spark and Apache Hadoop From On-Premises to AWS

Originally published December 2, 2020

Updated August 22, 2022



Notices

Customers are responsible for making their own independent assessment of the information in this document. This document: (a) is for informational purposes only, (b) represents current AWS product offerings and practices, which are subject to change without notice, and (c) does not create any commitments or assurances from AWS and its affiliates, suppliers or licensors. AWS products or services are provided “as is” without warranties, representations, or conditions of any kind, whether express or implied. The responsibilities and liabilities of AWS to its customers are controlled by AWS agreements, and this document is not part of, nor does it modify, any agreement between AWS and its customers.

Contents

About this Guide

For many customers, migrating to Amazon EMR raises many questions about assessment, planning, architectural choices, and how to meet the many requirements of moving analytics applications like Apache Spark and Apache Hadoop from on-premises data centers to a new AWS Cloud environment. Many customers have concerns about the viability of distribution vendors or a purely open-source software approach, and they need practical advice about making a change. This guide includes the overall steps of migration and provides best practices that we have accumulated to help customers with their migration journey.

Overview

Businesses worldwide are discovering the power of new big data processing and analytics frameworks like Apache Hadoop and Apache Spark, but they are also discovering some of the challenges of operating these technologies in on-premises data lake environments. Not least, many customers need a safe long-term choice of platform as the big data industry is rapidly changing and some vendors are now struggling.

Common problems include a lack of agility, excessive costs, and administrative headaches, as IT organizations wrestle with the effort of provisioning resources, handling uneven workloads at large scale, and keeping up with the pace of rapidly changing, community-driven, open-source software innovation. Many big data initiatives suffer from the delay and burden of evaluating, selecting, purchasing, receiving, deploying, integrating, provisioning, patching, maintaining, upgrading, and supporting the underlying hardware and software infrastructure.

A subtler, if equally critical, problem is the way companies' data center deployments of Apache Hadoop and Apache Spark directly tie together the compute and storage resources in the same servers, creating an inflexible model where they must scale in lock step. This means that almost any on-premises environment pays for high amounts of under-used disk capacity, processing power, or system memory, as each workload has different requirements for these components.

How can smart businesses find success with their big data initiatives?

Migrating big data (and machine learning) to the cloud offers many advantages. Cloud infrastructure service providers, such as Amazon Web Services (AWS), offer a broad choice of on-demand and elastic compute resources, resilient and inexpensive persistent storage, and managed services that provide up-to-date, familiar environments to develop and operate big data applications. Data engineers, developers, data scientists, and IT personnel can focus their efforts on preparing data and extracting valuable insights.

Services like Amazon EMR, AWS Glue, and Amazon S3 enable you to decouple and scale your compute and storage independently, while providing an integrated, well-managed, highly resilient environment, immediately reducing so many of the problems of on-premises approaches. This approach leads to faster, more agile, easier to use, and more cost-efficient big data and data lake initiatives.

However, the conventional wisdom of traditional on-premises Apache Hadoop and Apache Spark isn't always the best strategy in cloud-based deployments. A simple *lift and shift* approach to running cluster nodes in the cloud is conceptually easy but suboptimal in practice. Different design decisions go a long way towards maximizing your gains as you migrate big data to a cloud architecture.

This guide provides the best practices for:

- Migrating data, applications, and catalogs
- Using persistent and transient resources

- Configuring security policies, access controls, and audit logs
- Estimating and minimizing costs, while maximizing value
- Leveraging the AWS Cloud for high availability (HA) and disaster recovery (DR)
- Automating common administrative tasks

Although not intended as a replacement for professional services, this guide covers a wide range of common questions, and scenarios as you migrate your big data and data lake initiatives to the cloud.

Starting Your Journey

Migration Approaches

When starting your journey for migrating your big data platform to the cloud, you must first decide how to approach migration. One approach is to *re-architect* your platform to maximize the benefits of the cloud. The other approach is known as *lift and shift*, is to take your existing architecture and complete a straight migration to the cloud. A final option is a hybrid approach, where you blend a lift and shift with re-architecture. This decision is not straightforward as there are advantages and disadvantages of both approaches.

A lift and shift approach is usually simpler with less ambiguity and risk. Additionally, this approach is better when you are working against tight deadlines, such as when your lease is expiring for a data center. However, the disadvantage to a lift and shift is that it is not always the most cost effective, and the existing architecture may not readily map to a solution in the cloud.

A re-architecture unlocks many advantages, including optimization of costs and efficiencies. With re-architecture, you move to the latest and greatest software, have better integration with native cloud tools, and lower operational burden by leveraging native cloud products and services.

This paper provides advantages and disadvantages of each migration approach from the perspective of the Apache Hadoop ecosystem. For a general resource on deciding which approach is ideal for your workflow, see An E-Book of Cloud Best Practices for Your Enterprise, which outlines the best practices for performing migrations to the cloud at a higher level.

Re-Architecting

Re-architecting is ideal when you want to maximize the benefits of moving to the cloud. Re-architecting requires research, planning, experimentation, education, implementation, and deployment. These efforts cost resources and time but generally provide the greatest rate of return as reduced hardware and storage costs, operational maintenance, and most flexibility to meet future business needs.

A re-architecture approach to migration includes the following benefits for your applications:

- Independent scaling of components due to separated storage and compute resources.
- Increased productivity and lowered costs by leveraging the latest features and software.
- Ability to prototype and experiment quickly due to provisioning resources quickly.
- Options to scale system vertically (by requesting more powerful hardware) and horizontally (by requesting more hardware units).
- Lowered operational burden by no longer managing many aspects of cluster lifecycle, including replacing failed nodes, upgrades, patching, etc. Since clusters can be treated as transient resources, they can be decommissioned and restarted.

- Data accessibility when using a data lake architecture, data is stored on a central storage system that can be used by a wide variety of services and tools to ingest and process the data for different use cases. For example, using services such as AWS Glue, and Amazon Athena and other services can greatly reduce operational burden and reduce costs, and can only be leveraged if data is stored on S3.
- Ability to treat compute instances as transient resources, and only use as much as you need, when you actively need it.

Best Practices for Re-architecting

When re-architecting your system for the use of Amazon EMR, consider the following best practices:

- Read the documentation found in this guide for reference architectures and approaches that others have taken to successfully migrate.
- Reach out to an AWS representative early for a roadmap on architectures that would meet your use case and goals.

Lift and Shift

The lift and shift approach is the ideal way of moving workloads from on-premises to the cloud when time is critical and ambiguity is high. Lift and shift is the process of moving your existing applications from one infrastructure to another. The benefits to this approach are:

- Fewer number of changes. Since the goal is to move applications to environments that are similar to the existing environment, changes are limited to only those required to make the application to work on the cloud.
- Less risk because fewer changes reduce the unknowns and unexpected work.
- Shorter time to market because fewer number of changes reduces the amount of training needed by engineers.

Best Practices for Lift and Shift

- Consider using Amazon S3 for your storage instead of HDFS because this approach reduces costs significantly, and it allows you to scale compute to the amount of data. When using HDFS, the data must be replicated by at least two times, requiring more storage. The main cost driver is the cost of storage by storing the data on EC2 instances using expensive disk-based instances or using large EBS volumes. A quick calculation using AWS cost calculator shows that storage costs for HDFS can be up to three times the cost of Amazon S3. See Using Amazon S3 as the Central Data Repository for a guide.
- Amazon EMR bundles several versions of applications to a single Amazon Machine Image (AMI) that you choose. If you choose newer versions of an application, research the changes that were made between versions and look for known issues. In cases of open-source applications, upgrading to a newer version may introduce bugs.

- Amazon EMR clusters are configured using defaults that depend on the instance types chosen. See Task Configuration for default values at the Hadoop task level and Spark Defaults Set by Amazon EMR for Apache Spark defaults. These defaults run for most workloads but some jobs may require that you override these defaults.
- Amazon EMR clusters by default use the capacity scheduler as the Apache Hadoop resource scheduler. Validate that this scheduler fits the use case that you are migrating.

Hybrid Architecture

Hybrid architectures leverage aspects of both lift and shift and re-architecting approaches. For existing applications, a lift and shift approach is employed for a quick migration. Any new applications then can use re-architected architecture. This hybrid approach includes the benefit of being able to experiment and gain experience with cloud technologies and paradigms before moving to the cloud.

Prototyping

When moving to a new and unfamiliar product or service, there is always a period of learning. Usually, the best way to learn is to prototype and learn from doing, rather than researching alone, to help identify the unknowns early in the process so you can plan for them later. Make prototyping mandatory to challenge assumptions. Common assumptions when working with new products and services include the following:

A particular data format is the best data format for my use case. Many times, customers read that a specific data format is the best. However, there is no best data format for all use cases. There are data formats that perform better than most other data formats, such as Apache Parquet and Apache ORC. Validating this assumption could be relatively quick but the impact could be large. Switching data formats after moving to production is more expensive than running a quick prototype with real-world data. This prototyping is highly recommended.

A particular application is more performant than another application for processing a specific workflow. This scenario is the same as the above common assumption on data formats. Changing applications can be an expensive undertaking later in the process and impact adoption.

A particular instance type is the most cost effective way to run a specific workflow. Many times, another instance type performs better if it is tuned for the workflow. For example, C series EC2 instances can perform better, and cost less if you enable spill-to-disk rather than using R series EC2 instances. This scenario is easier to change later on and is recommended if cost and performance is a high priority requirement.

A particular application running on-premises should work identically on cloud. There are many factors that contribute to running workloads, such as the instance type, storage type, application version, infrastructure configuration, and so on. Running a wide variety of jobs with real data that you expect to run on production provides the most validation.

With the cloud, there are several factors in the environment in which a workload may run. For example, at different times of day, traffic to Amazon S3 could vary, or caching could be instituted when not expected. Therefore, prototyping reduces the number and severity of surprises during development and deployments, and the rate of return can be large. Last, finding out issues earlier than later in the development cycle is much more cost effective.

Best Practices for Prototyping

- Decide on what to prototype, brainstorm all of the possible assumptions and unknowns. Make the assumptions and unknowns with the largest potential impact a priority.
- Start early and choose the riskiest aspects to prototype first.
- Prototype in an environment that is similar to the environment that you want to be operating in. Start with a smaller environment or subset of characteristics and then move to a larger scale.
- Determine goals for the tests upfront and get support from stakeholders. The goal could be to answer a question about how something works or to validate a design.
- Make tests deterministic and easily repeatable. Run experiments using an automated approach, such as a script or continuous integration environment, so that the test can be run in different environments. For example, run a test on different instance types or against multiple AMIs. These scripts can later be used as tests for deployments.
- Validate your test setup, environment, and results with someone else to ensure that all factors are being considered. For example, if you run download tests against the same S3 objects, this could cause Amazon S3 to cache the object. This scenario gives incorrect results when the actual workflow is retrieving random objects.
- Run the tests sufficiently enough to remove variability that may come from dependencies. For example, variability from Amazon S3 may be caused by the traffic load of other users or the time of day. Look at different percentiles, such as P50, P90, P99, and P100 and determine how variability may impact user experience.
- Document the results and have them reviewed by your team members. This review ensures that the tests were run properly and results are consistent.
- Don't make assumptions. In the big data analytics space, too many variables affect performance, cost, and functionality meaning an obvious assumption may be incorrect. Always validate your assumptions by testing them. For example, many people may assume that a particular instance type that closely matches the hardware they use on their premises may be a better fit than choosing another instance type.

- The purpose of prototyping is to validate assumptions of a design or approach with a high degree of certainty. Generally, the more effort put into a prototype by accounting for multiple factors will yield higher certainty that the design will work in a production setting. Determining your goals at the beginning of the process will make sure that you stop at a reasonable level.
- Don't be afraid to seek help by posting on forums, consulting AWS partners, and reaching out to your account team.

Choosing a Team

When starting a migration to the cloud, you must carefully choose your project team to research, design, implement, and maintain the new system. We recommend that your team has individuals in the following roles with the understanding that a person can play multiple roles:

- A **project leader** capable of managing the project end-to-end, and with enough technical expertise and background on big data technology to make decisions.
- A **big data applications engineer** with a strong background in Apache Hadoop and other technologies that are being migrated. This background is helpful to understand how the underlying applications work, their intended uses, and their limitations.
- An **infrastructure engineer** well-versed in the automation of infrastructure provisioning on AWS and familiar with tools like AWS CloudFormation, Terraform (by HashiCorp), and so on. This person should also be familiar with test automation, including CI/CD approaches and tools.
- A **security engineer** able to provide guidance on the security requirements that your company mandates. This person should be knowledgeable enough to map the organization's security requirements to security features offered by AWS. This person should also be flexible about the security design as long as it meets the intended requirements.
- A **group of engineers** who are quick learners and are not afraid to dive into areas that they may not be familiar with.

All members in the migration team must exhibit the following characteristics:

- They must have an open mind and ability to think differently. Cloud infrastructure requires a paradigm shift on how resources are treated.
- They must be able to dive deep into the underlying frameworks, architectures, and issues.
- They must share an agreement on the project's mission and direction.

General Best Practices for Migration

Migrating big data and analytics workloads from on-premises to the cloud involves careful decision making. The following are general best practices to consider when migrating these workloads to Amazon EMR:

Consider using AWS Glue, Amazon Redshift, or Amazon Athena. Although Amazon EMR is flexible and provides the greatest amount of customization and control, there is an associated cost of managing Amazon EMR clusters, upgrades, and so on. Consider using other managed AWS services that fulfill your requirements as they may have lower operational burden, and in some cases, lower costs. If one of these services does not meet your use case requirements, then use Amazon EMR.

Use Amazon S3 for your storage (data lake infrastructure). A data lake is a centralized repository that allows you to store all of your structured and unstructured data at any scale. Data can be stored as-is, without having to first structure the data. You can execute different types of analytics on the data, from dashboards and visualizations to big data processing, real-time analytics, and machine learning.

Amazon S3 is architected for high durability and high availability and supports lifecycle policies for tiered storage. For more details, see Using Amazon S3 as the Central Data Repository.

Decouple compute from storage so that you can scale them independently. With your data in Amazon S3, you can launch as much or as little compute capacity as you need using Amazon Elastic Compute Cloud (EC2). For more information, see Benefits of using Amazon S3.

Use multiple Amazon EMR (transient and long running) clusters with the ability to spin up and down on demand. Analyze the current workloads and assign the workloads to different clusters based on usage patterns.

- Separate out batch jobs (extract, transform, load [ETL], aggregations, data cleansing, roll-up, machine learning) and interactive queries (one-time analysis).
- Use Reserved and Spot Instances as applicable to reduce costs for baseline or variable workloads, respectively.
- Use automatic scaling within clusters. Automatic scaling allows for programmatically scaling in and out core nodes and task nodes based on Amazon CloudWatch metrics and other parameters that are specified in a scaling policy.
- Right-size clusters (both instance types and number of instances). Multiple Amazon EC2 instance types are available—make sure to choose the correct instance type based on the workload. For more details, see Cost Estimation and Optimization.

Implement automation and continuous integration/continuous delivery (CI/CD) practices to enable experimentation and efficiency. Automating the provisioning of EMR clusters along with other resources like roles and security groups is an operational excellence best practice. Apply the same engineering discipline to infrastructure that is typically used for application code. Check the infrastructure code into a code repository and build CI/CD pipelines to test the code. Implementing

infrastructure as code also allows for the provisioning of EMR clusters in another Availability Zone or AWS Region should problems arise in the one currently being used. For more details, see Operational Excellence.

Involve security and compliance engineers as early in the migration process as possible and make sure that the EMR environments are in line with the organization's security directives. Make full use of multiple security-related services, such as AWS Identity and Access Management (IAM) and AWS Key Management Service (KMS), and features, such as Security Configurations within EMR. Amazon S3 also includes many security-related features. Make sure that all data is encrypted at-rest and in-transit. Finally, make sure that authentication and authorization are enabled as appropriate. For more details, see Securing your Resources on Amazon EMR.

Gathering Requirements

Obtaining On-Premises Metrics

The following list of metrics is useful to help with cost estimation, architecture planning, and instance type selection.

Capture each of these metrics on your existing Hadoop clusters to help drive the decision-making process during migration.

- Aggregate number of physical CPUs
- CPU clock speed and core counts
- Aggregate memory size
- Amount of HDFS storage (without replication)
- Aggregate maximum network throughput
- At least one week of utilization graphs for the resources used above

For help with taking a full inventory of your on-premises architecture and the possible requirements for migration, refer to Appendix A: Questionnaire for Requirements Gathering.

Cost Estimation and Optimization

With Amazon EMR, you only pay a per-second rate for every second that you use the cluster. The cost is based on the instance type, the number of Amazon EC2 instances that you deploy, and the AWS Region in which you launch your cluster. The net cost includes Amazon EMR cost in addition to the Amazon EC2 cost and Amazon EBS cost (if using EBS volumes), which are also billed per-second. This section details how you can optimize Amazon EMR clusters and leverage features such as Auto Scaling and Instance Fleets to lower costs.

Optimizing Costs

Amazon EMR provides multiple features to help lower costs. To best utilize these features, consider the following factors.

Workload Type

You can run different applications and workload types on Amazon EMR. For applications that only run for a specific period, you can use a *transient* EMR cluster. For applications that run for a long period, you can use a *long-running* cluster. The following image shows typical workload types and whether they're classified as transient or long-running.

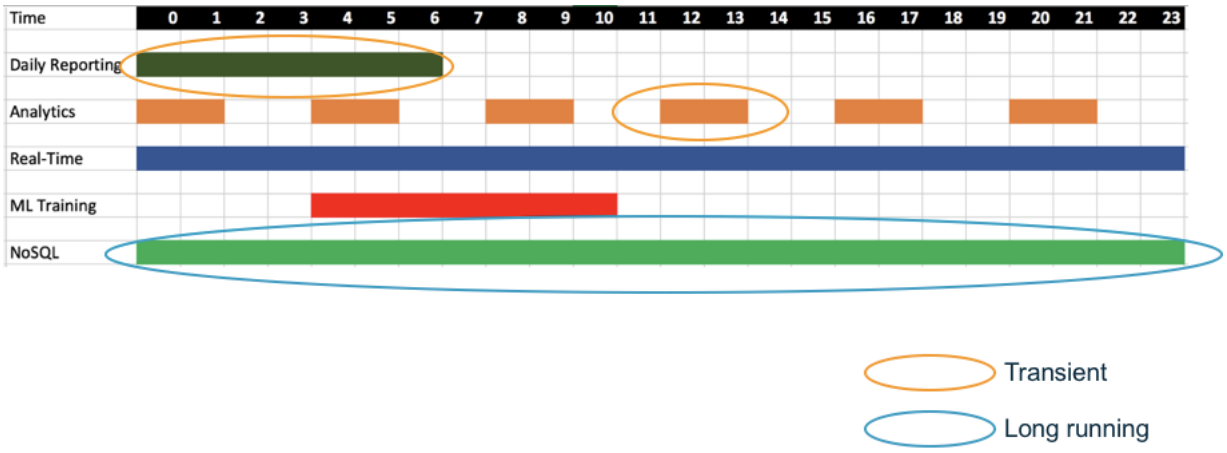


Figure 1: Typical workloads and their cluster types

Instance Type

Most Amazon EMR clusters can run on general-purpose EC2 instance types/families such as m4.xlarge and m5.xlarge. Compute-intensive clusters may benefit from running on high performance computing (HPC) instances, such as the compute-optimized instance family (C5). Database and memory-caching applications may benefit from running on high memory instances, such as the memory-optimized

instance family (R5). The primary node does not have large computational requirements. For most clusters of 50 or fewer nodes, you can use a general-purpose instance type such as m5.xlarge.

> **Note:** For clusters of more than 50 nodes, consider using a larger instance type, such as m4.xlarge). The amount of data you can process depends on the capacity of your core nodes and the size of your data as input, data during processing, and data as output.

Application Settings

Job performance also depends on application settings. There are different application settings for different use cases. For example, by default, EMR clusters with Apache HBase installed allocate half of the memory for HBase and allocate the other half of memory for Apache Hadoop YARN. If you use applications such as Apache HBase and Apache Spark, we recommend that you don't use a single, larger cluster for both applications. Instead run each application on a separate, smaller cluster.

Storage Optimization

When you use Amazon EMR, you have the ability to decouple your compute and storage by using Amazon S3 as your persistent data store. By optimizing your storage, you can improve the performance of your jobs. This approach enables you to use less hardware and run clusters for a shorter period. Here are some strategies to help you optimize your cluster storage for Amazon S3:

Partition Data

When your data is partitioned and you read the data based on a partition column, your query only reads the files that are required. This reduces the amount of data scanned during the query. For example, the following image shows two queries executed on two datasets of the same size. One dataset is partitioned, and the other dataset is not.

```
presto:costdb> SELECT count(*) FROM "costdb"."s3logsjsonpartitioned"
            -> WHERE "year" = 2018
            -> AND "month" BETWEEN 3 AND 5
            -> AND "day" != 14;
 _col0
--------
 468006
(1 row)

Query 20181109_203220_00004_adurx, FINISHED, 3 nodes
Splits: 4,323 total, 4,323 done (100.00%)
0:20 [468K rows, 349MB] [22.9K rows/s, 17.1MB/s]

presto:costdb> SELECT count(*) FROM "costdb"."s3logsjsonnopartition"
            -> WHERE "year" = 2018
            -> AND "month" BETWEEN 3 AND 5
            -> AND "day" != 14;
 _col0
--------
 468006
(1 row)

Query 20181109_203328_00005_adurx, FINISHED, 3 nodes
Splits: 23,617 total, 23,617 done (100.00%)
2:48 [6M rows, 5.13GB] [35.7K rows/s, 31.2MB/s]
```

Figure 2: Queries on partitioned and non-partitioned data

The query over the partitioned data (`s3logsjsonpartitioned`) took 20 seconds to complete and it scanned 349 MB of data.

The query over the non-partitioned data (`s3logsjsonnopartition`) took 2 minutes and 48 seconds to complete and it scanned 5.13 GB of data.

Optimize File Sizes

Avoid files that are too small (generally, anything less than 128 MB). By having fewer files that are larger, you can reduce the amount of Amazon S3 LIST requests and also improve the job performance. To show the performance impact of having too many files, the following image shows a query executed over a dataset containing 50 files and a query over a dataset of the same size, but with 25,000 files.

```
presto:costdb> select count(*) from fewfilesjson;
   _col0
-----------
 200000012
(1 row)

Query 20181108_230835_00008_5ma32, FINISHED, 3 nodes
Splits: 567 total, 567 done (100.00%)
2:31 [200M rows, 25.6GB] [1.33M rows/s, 174MB/s]

presto:costdb> select count(*) from manyfilesjson;
   _col0
-----------
 200000012
(1 row)

Query 20181108_232128_00009_5ma32, FINISHED, 3 nodes
Splits: 25,017 total, 25,017 done (100.00%)
3:47 [200M rows, 25.6GB] [882K rows/s, 116MB/s]
```

Figure 3: Query time difference on number of files

The query executed over the dataset containing 50 files (`fewfilesjson`) took 2 minutes and 31 seconds to complete. The query over the dataset with 25,000 files (`manyfilesjson`) took 3 minutes and 47 seconds to complete.

Compress the Dataset

By compressing your data, you reduce the amount of storage needed for the data, and minimize the network traffic between S3 and the EMR nodes. When you compress your data, make sure to use a compression algorithm that allows files to be split or have each file be the optimal size for parallelization on your cluster. File formats such as Apache Parquet or Apache ORC provide compression by default. The following image shows the size difference between two file formats, Parquet (has compression enabled) and JSON (text format, no compression enabled). The Parquet dataset is almost five times smaller than the JSON dataset:

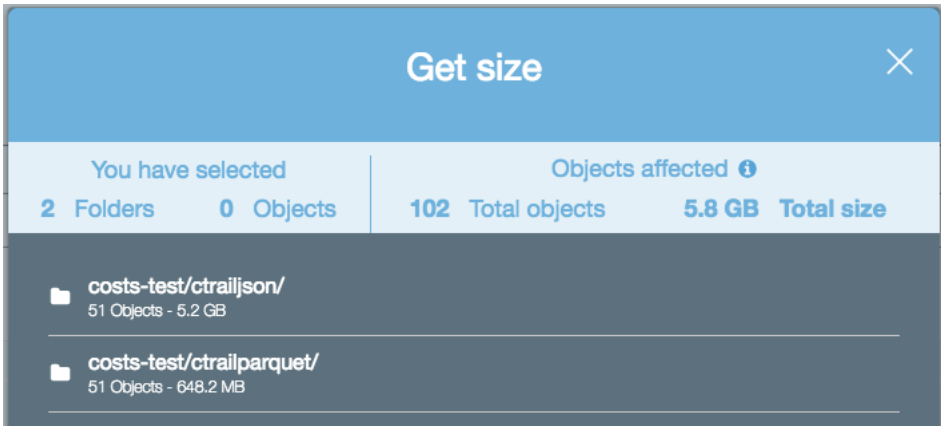


Figure 4: Compressed and non-compressed datasets

Optimize File Formats

Columnar file formats like Parquet and ORC can improve read performance. Columnar formats are ideal if most of your queries only select a subset of columns. For use cases where you primarily select all columns, but only select a subset of rows, choose a row-optimized file format such as Apache Avro. The following image shows a performance comparison of a `select count(*)` query between Parquet and JSON (text) file formats.

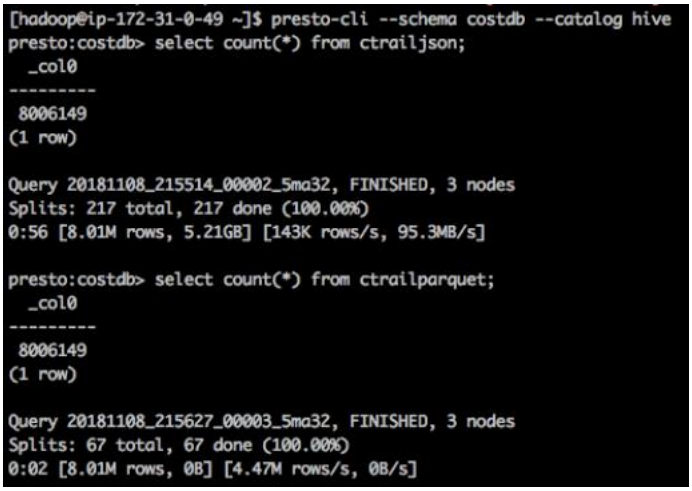


Figure 5: Performance comparison of file formats

The query over the JSON dataset took 56 seconds to complete, and it scanned 5.21 GB of data. The query over the Parquet dataset took 2 seconds to complete, and in this example, it did not need to scan any data.

Compute Optimization

In the previous section, we covered some of the strategies that you can use to optimize your Amazon S3 storage costs and performance. In this section, we cover some of the features and ways to optimize your Amazon EC2 cluster's compute resources.

Amazon EC2 provides various purchasing options. When you launch Amazon EMR clusters, you have the ability use On-Demand, Spot, or Reserved EC2 instances. Amazon EC2 Spot Instances offer spare compute capacity available at discounts compared to On-Demand Instances. Amazon EC2 Reserved Instances enable you to reserve EC2 instances at a significant discount compared to On-Demand pricing. For more detailed information, see Instance Purchasing Options in the *Amazon EMR Management Guide*.

Spot Instances

Running EMR clusters with Spot instances can be useful for a number of scenarios. However, there are few things that you must consider before choosing Spot instances for a particular workload. For

example, if you're running a job that requires predictable completion time or has service level agreement (SLA) requirements, then using Spot instances may not be the best fit. For workloads where they can be interrupted and resumed (interruption rates are extremely low), or workloads that can exceed an SLA, you can use Spot instances for the entire cluster.

You can also use a combination of Spot and On-Demand instances for certain workloads. For example, if cost is more important than the time to completion, but you cannot tolerate a partial loss of work (have an entire cluster terminated), you can use Spot instances for the task nodes, and use On-Demand/Reserved instances for the primary and core nodes.

Spot instances are also great for testing and development workloads. You can use Spot instances for an entire testing cluster to help you reduce costs when testing new applications.

Reserved Instances

With Reserved Instances (RIs), you can purchase/reserve EC2 capacity at a lower price compared to On-Demand Instances. Keep in mind that for you to have reduced costs with RIs, you must make sure that your RI use over a period of a year is higher than 70%. For example, if you use transient EMR clusters and your clusters only run for a total of 12 hours per day, then your yearly use is 50%. This means that RIs might not help you reduce costs for that workload. Reserved Instances may help you to reduce costs for long-running clusters and workloads.

Savings Plans

Savings Plans is a flexible discount model that provides you with the same discounts as Reserved Instances, in exchange for a commitment to use a specific amount (measured in dollars per hour) of compute power over a one- or three-year period. Every type of compute usage has an On-Demand price and a (lower) Savings Plan price. After you commit to a specific amount of compute usage per hour, all usage up to that amount will be covered by the Saving Plan, and anything past it will be billed at the On-Demand rate. If you have Reserved Instances, the Savings Plan applies to any On Demand usage that is not covered by the RIs. Savings Plans are available in two options:

- **Compute Savings Plans** provide the most flexibility and help to reduce your costs by up to 66%. The plans automatically apply to any EC2 instance regardless of region, instance family, operating system, or tenancy, including those that are part of your EMR clusters.
- **EC2 Instance Savings Plans** apply to a specific instance family within a region and provide the largest discount (up to 72%, just like Standard RIs). Like RIs, your savings plan covers usage of different sizes of the same instance type (such as a c5.4xlarge or c5.large) throughout a region.

Instance Fleets

Instance fleets is an Amazon EMR feature that provides you with variety of options for provisioning EC2 instances. This approach enables you to easily provision an EMR cluster with Spot Instances, On-Demand

Instances, or a combination of both. When you launch a cluster with Instance Fleets, you can select the target capacity for On-Demand and Spot Instances and specify a list of instance types and Availability Zones. Instance fleets choose the instance type and the Availability Zone that is the best fit to fulfill your launch request.

Instance fleets also provide many features for provisioning Spot Instances. This includes the ability for you to specify a defined duration (Spot block) to keep the Spot Instances running, the maximum Spot price that you're willing to pay, and a timeout period for provisioning Spot Instances.

Amazon EMR Auto Scaling

You can reduce costs by using the Amazon EMR automatic scaling feature to dynamically scale your cluster. This feature enables you to select Amazon CloudWatch metrics; you can also specify scaling policies to automatically scale out (add more nodes to the cluster) and scale in (remove nodes from the cluster) core nodes and task nodes. This approach enables you to run EMR clusters with just the correct of amount of resources that your application needs. This feature is also useful for use cases where you have spikes in cluster utilization (i.e. a user submitting a job) and you want the cluster to automatically scale based on the requirements for that application. For more information, see Best practices for resizing and automatic scaling in Amazon EMR on the *AWS Big Data Blog*.

Cost Estimation Summary

There are a number of factors to consider when estimating costs for an Amazon EMR cluster. These factors include EC2 instances (compute layer), Amazon Elastic Block Stores (Amazon EBS) volumes, and Amazon S3 storage. Due to the per-second pricing of Amazon EMR, the cost of running a large EMR cluster that runs for a short duration would be similar to the cost of running a small cluster for a longer duration. For example, a 10-node cluster running for 10 hours costs the same as a 100-node cluster running for 1 hour. The hourly rate depends on the instance type used (such as standard, high CPU, high memory, high storage, etc.). For detailed pricing information, see Amazon EMR Pricing.

Optimizing Apache Hadoop YARN-based Applications

About Apache Hadoop YARN and Job Optimization

Apache Hadoop YARN is the resource management and job scheduling technology in the open source Hadoop distributed processing framework. Nodes are registered with YARN and provide virtual memory and virtual CPU in which the cluster uses to process jobs. The total resources available to a cluster is the sum of all of the nodes that participate in job processing. Ideally, the YARN memory and CPU resources should closely match those of the underlying hardware, but in some cases, adjustments are needed. With Amazon EMR, defaults are provided for each node. For more information on these defaults, see Task Configuration in the *Amazon EMR Release Guide*.

A job requires resources to complete its computation. To parallelize the job, YARN runs subsets of work within containers called tasks. The job requests from YARN the amount of memory and CPU expected for each container. If not specified, then a default container size is allocated for each container. If the container is not sized properly, the job may waste resources because it's not using everything allocated to it, run slowly because it's constrained, or fail because its resources are too constrained.

To ensure that the underlying hardware is fully utilized, you must take into consideration both the resources in YARN and the requests coming from a job. YARN manages virtual resources but does not necessarily map to the underlying hardware. In addition, YARN configuration and task schedule configuration does have an impact on how the underlying hardware is used.

Optimizing and Monitoring Your Cluster

To tune your cluster, first ensure that YARN is optimized. If some containers are constantly available, shrinking your cluster saves cost without decreasing performance because containers are sitting idle. Amazon EMR emits a ContainerPending metric to Amazon CloudWatch that can provide this information. If there is a constant queue of container requests, then increasing your cluster size helps your applications finish faster because they can benefit from increased parallelism.

To ensure that you are using all of the physical resources, use Ganglia monitoring software to provide information about the underlying hardware. If 100% of YARN resources (vCPU and Memory) are used, but Ganglia is showing that actual CPU and memory usage is not crossing 80%, then you may want to reduce container size so that the cluster can run more concurrent containers.

If looking at Ganglia shows that either CPU or memory is 100% but the other resources are not being used significantly, then consider moving to another instance type that may provide better performance at a lower cost. For example, if CPU is 100%, and memory usage is less than 50% on R4 or M5 series instance types, then moving to C4 series instance type may be able to address the bottleneck on CPU.

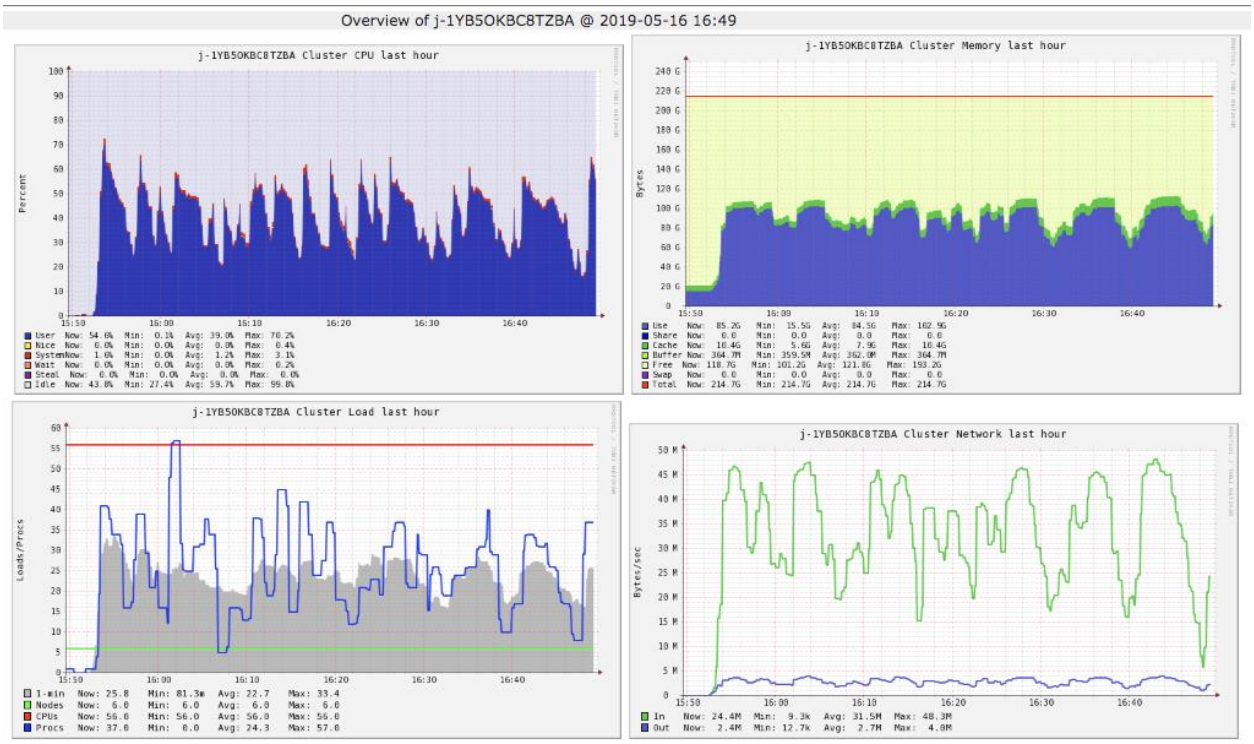


Figure 6: Example Ganglia graph showing underutilized CPU and memory

Tuning Controls for Cluster Optimization

If CPU and memory are not fully utilized, you can tune two controls. The first control is the amount of resources each container uses. To tune the resources available for a container, you can change the default container sizes using Amazon EMR Application Configuration. The same configurations can be used to control memory and CPU at an application level to provide finer control of resources. In general, you want to ensure that the size of your containers are a multiple of the total amount of resources allocated to YARN.

The second control is the amount of virtual resources that is reservable from each node within YARN. To change the amount of YARN memory or CPU available to be reserved on each node in your cluster, set the `yarn.nodemanager.resource.memory-mb` and `yarn.nodemanager.resource.cpu-vcores` configurations using the Amazon EMR configuration API. For default values, see Hadoop Daemon Configuration Settings in the Amazon EMR Release Guide.

The following decision graph provides a suggested approach to optimizing your jobs.

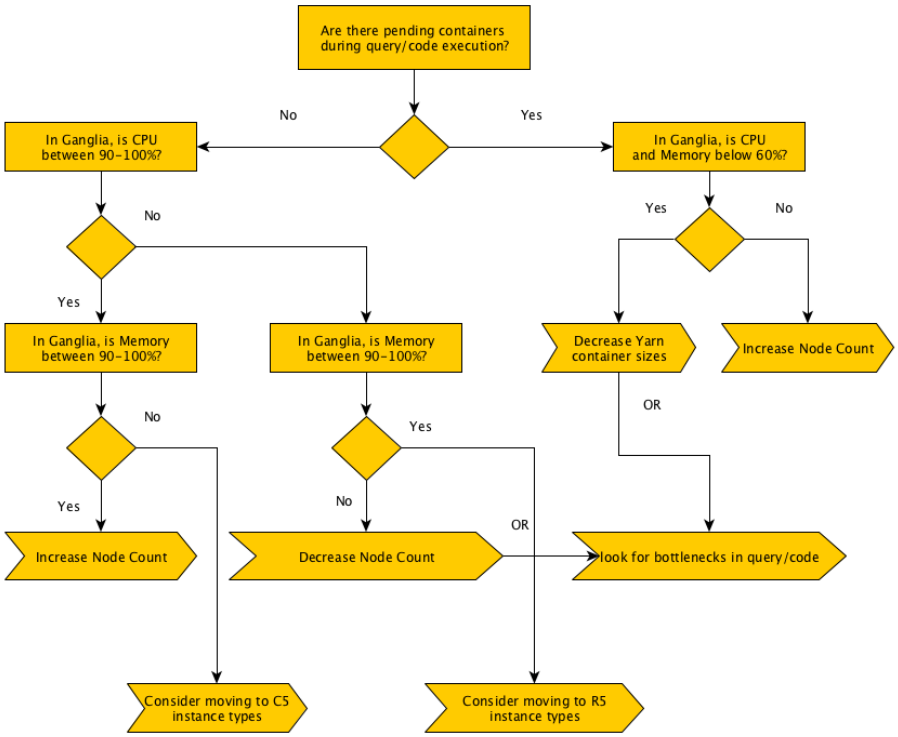


Figure 7: Job optimization decision chart

Amazon EMR Cluster Segmentation Schemes

One of the main benefits of using Amazon EMR is that it makes it easy for you to start and terminate clusters. Starting and stopping clusters quickly gives you the flexibility that a single, long-running cluster cannot provide, and provides opportunities to save costs by leveraging Amazon EC2 Spot Instances.

This section covers a few approaches to splitting a single, permanently running cluster into smaller clusters and identifies the benefits these practices bring to your environments. The approach you choose depends on your use case and existing workflows. AWS can work with you to help choose the strategy that meets your goals.

Cluster Characteristics

You can approach the task of splitting up existing cluster from different perspectives, depending on the area or a set of characteristics that you want to tackle or address. The strategy you choose depends on the scenarios you have and the goals you want to achieve. The following cluster characteristics are typically considered.

Machine Types and Cluster Sizes

In your cluster environment, different established workflows may experience bottlenecks on different resources, such as the number of CPUs, memory size, network bandwidth, or network latency. In small clusters that you start on demand, you select machine types to best match the workloads. Similarly, you select the amount of resources your clusters need. For example, you can use high-memory instances for processing machine learning algorithms, or run compute-optimized instances for query engines, like PrestoDB.

Applications, Application Versions, and Configuration

You can create different configurations and deploy different applications on different clusters, providing only those resources that users need. You can also create clusters that allow you to run red-black testing during application version upgrades, and to test new software.

Security

IAM Roles Specific to Each Smaller Cluster

Amazon EMR clusters are assigned an EC2 role that provides permissions to access different AWS resources. You can assign different roles to your clusters so that these roles have different access and permissions to resources, such as Amazon S3 buckets or Amazon Kinesis Data Streams. This role assignment allows you to provide carefully restricted permissions to each EMR cluster, preventing unintended actions and reducing the scope of risk if any security events occur.

Security Controls
Depending on the use cases that a cluster serves, you can change the level of security for the cluster. For example, if one cluster is used purely for batch processing that is submitted from a workflow engine or via steps, securing this cluster at a user level may not be a top priority. In this case, forgoing Kerberos configuration may be acceptable since no one is interacting with the cluster. For a cluster that serves interactive use cases, you may need to configure Kerberos, restrict SSH access, and follow other security practices. For information on security controls available in Amazon EMR, see Security in the *Amazon EMR Management Guide*.

Network Controls
You can assign different clusters to different security groups or place them in different subnets that control access from specified network sources.

Disaster recovery

Using more than one cluster provides redundancy to minimize impact if a single cluster goes down or is taken offline. The following examples are a couple use cases where having multiple clusters can help:

- A cluster becomes unhealthy due to a software bug, network issue, or other external dependencies being unavailable.
- A maintenance operation needs to occur such as a software upgrade, patching that requires a machine reboot, or bouncing of applications.

Common Cluster Segmentation Schemes

Lifecycle Stages

One of the typical approaches to deciding how to segregate clusters is based on having dedicated clusters for separate stages in your lifecycle, such as testing, beta, and production. This way, jobs that are not ready for production can run on their own dedicated cluster and do not interfere or compete with production jobs for resources or writing of data results. Having different clusters for separate stages also lets you test jobs on clusters that have newer versions of applications. This approach lets you test upgrades before upgrading your beta or production environments. To further isolate your workflows and scenarios, you can apply a separate instance role in Amazon EMR that disallows beta jobs to write their results to production S3 locations, protecting them from accidental deletions or modifications arising from your beta stage environment.

Workload Types

Clusters that serve end users who submit ad hoc queries typically require stricter security controls. They also must use different applications, such as Apache Hue and Apache Zeppelin. These interactive workload clusters usually have peak usage times during business hours and low usage at all other times.

Configuring automatic scaling policies for these clusters is beneficial, especially if you run them on Spot Instances.

Clusters used for batch/ETL workloads also tend to have peak usage times that are usually different from those of interactive workload clusters used for queries. Batch/ETL workload clusters can leverage automatic scaling so that workloads scale independently of other clusters and can scale up and down. For more information, see Using Automatic Scaling in Amazon EMR.

Time-Sensitive Jobs

Another common strategy for cluster segmentation is creating separate clusters based on whether their jobs are time-sensitive. When a repeated job's completion time must be consistent, creating and running the job on a separate cluster is a way to ensure that the job can obtain a predictable and consistent amount of resources each time that job must run. In addition, you can use more powerful and expensive hardware when running time-sensitive jobs.

Job Length

Running long jobs may consume available resources and take them away from other, shorter running jobs. Running separate clusters for long-running and short-running jobs can help short-running jobs complete faster, improve their SLA, and improve the SLA of workflows in general if they are in the critical path. Clusters that run short-running jobs also have a higher chance of completing jobs when Spot Instances are used because the chance of a job being interrupted by EC2 is lower.

Group/Organization Types

Some organizations create clusters per groups of users that share the same security requirements and ownership of EMR resources. This approach helps with restricting access to those groups, and also allows the administrators to allocate costs to the groups by using resource tagging. Having several, smaller clusters dedicated to separate groups also helps you isolate resources usage. With this approach, one group within your organization does not exhaust the resources of other groups using the same cluster. In addition, this approach reduces the risk that one group gains access to another group's data.

Additional Considerations for Segmentation

Segmenting your clusters too finely increases costs marginally. Add master nodes to a cluster to manage the cluster, although these instances do not need to be as powerful as the core and task nodes. In addition, adding more clusters adds complexity to your infrastructure because resources must be tracked. Users generally handle this complexity by using an orchestrator to ensure that resources are provisioned and shut down when appropriate, and build monitors to ensure that clusters are shut down when idle for a period of time. For a sample solution of this approach, see Optimize Amazon EMR costs

with idle checks and automatic resource termination using advanced Amazon CloudWatch metrics and AWS Lambda on the AWS Big Data Blog.

Finally, using multiple clusters also reduces the efficiency of instances, because they are not being shared by other jobs. However, this scenario can be offset by using automatic scaling.

Securing your Resources on Amazon EMR

Amazon EMR has a comprehensive range of tools and methods to secure your data processing in the AWS Cloud. This chapter provides information on Network Security, Authentication (proving identity), Authorization (granting identities to resources), Auditing (which identities accessed which resources when), Encryption, and Patching. At the end of this chapter, we provide examples of common customer setups that meet different use cases.

EMR Security Best Practices

Design early with security in mind. Implementing security designs at the beginning of migration saves time and reduces complexity because the architecture is built with security in mind. Large changes may require more effort when security becomes a requirement after implementation has been completed.

Ensure that the supporting department is involved early in security architecture. Have the department that reviews and approves architectures for security involved in the process as early as possible, and keep them up-to-date with decisions related to security. They may be able to give you advice earlier in the process to reduce or avoid design changes later in the process.

Understand the risks. Security is mainly about minimizing attack surfaces and minimizing impact should a system become compromised. No system can be entirely secured.

Obtain security exceptions. Security departments may provide security exceptions for rules that may no longer apply or where the risk of compromise is reduced. Getting security exceptions may significantly reduce the risk and scope of work needed to get approvals from a security department. For example, you may not need SELinux for Amazon EMR clusters that process batch jobs and in which there is no interactivity with users.

Use different security setups for different use cases. Batch and ETL clusters that do not have user interaction likely require a different security configuration than a cluster that is used in an interactive way. Clusters with interaction may have several users and processes that interact with a cluster and each user requiring different levels of access with each other. Clusters that are used for batch usually require much lower security controls than an interactive cluster.

Protect from unintentional network exposure. Security departments may configure proper security group rules to protect applications and data on the cluster. Misconfiguration of network security rules can open a broad range of cluster ports to unrestricted traffic from the public internet and expose cluster resources to outside threats. The Amazon EMR block public access feature allows you to minimize misconfigurations by centrally managing public network access to EMR clusters in an AWS Region. You can enable this configuration in an AWS Region and block your account users from launching clusters that allow unrestricted inbound traffic from the public IP address.

To learn more about EMR security best practices, see Best Practices for Securing Amazon EMR on the *AWS Big Data Blog*.

Authentication

Authentication is the process of an entity (a user or application) proving its identity to an application or system. When logging into an application or system, a user generally provides a login and password to prove that they are the user they are claiming to be. Other methods for authentication exist, such as providing certificates or tickets.

Authentication Between User and System

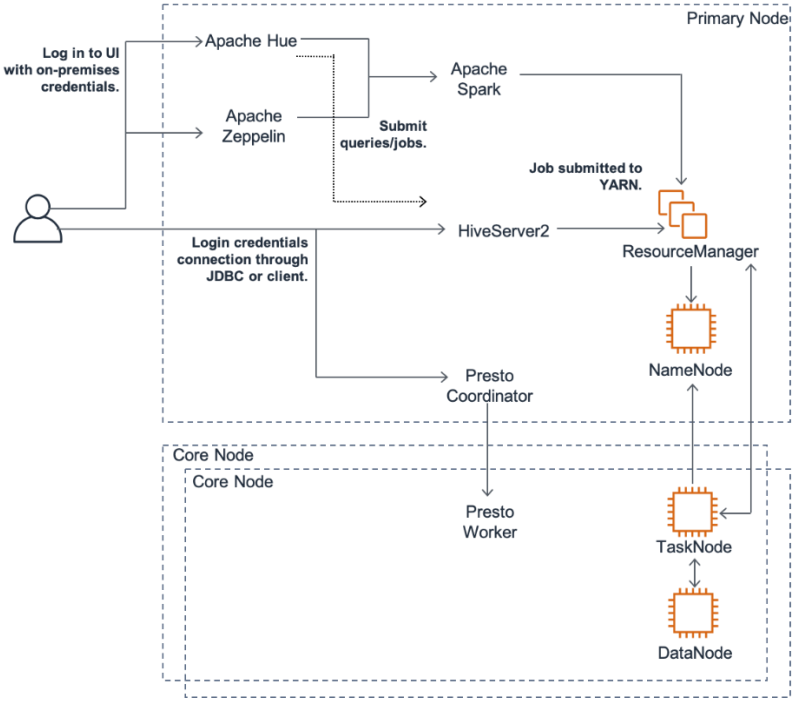


Figure 8: Authentication between user and system, applications

There are several ways to authenticate a user to an EMR cluster and/or applications on the Amazon EMR cluster.

SSH Keys as Authentication

An easier but less secure solution is to provide SSH keys to users that you want to have access to an Amazon EMR cluster. Those users with the keys can then SSH into the cluster and use any resources that the cluster has access to. With no other security features, those users that can log on to a cluster can run commands as root, and can access AWS resources that the Amazon EC2 instance role has access to.

This method is typically used when there are a small number of users or groups. Multiple EMR clusters can be started with different permissions and SSH keys, and the keys are distributed to users that have access to each cluster. Rotate the SSH keys periodically to ensure that the impact of leaked keys is limited. This is not a recommended approach if Amazon EMR clusters have access to sensitive data.

Lightweight Directory Access Protocol (LDAP) Authentication

Several applications support authentication against Active Directory or other directory services using lightweight directory access protocol (LDAP). With this method, users must provide the credentials that they normally use which are then sent to the on-premises Active Directory server for validation. Once these credentials are validated, the user then has access to the application.[1]

After a user is validated to an application, there are two ways a job can be submitted. In the first scenario, an application uses its own identity to submit a job for processing. For example, if Apache Hue is the application, then the Hue user submits the job and that job is run by the defined user *hue*. The second way to submit the job is through an application that impersonates the user to run that job. This method allows administrators to track usage of users and makes it easier to identify misuse of the system. By default, Apache Hue submits jobs impersonated by the user.

By default, Apache Hadoop's default authentication method is set to *simple*, which means that there is no user authentication. We recommend that you change this setting if the cluster is being used by multiple users. If Apache Hadoop is set up to use Kerberos and a YARN/HDFS job is submitted with Apache Spark or Apache Hive, then the end users must have an OS level account on each node as YARN/HDFS. In some cases, these accounts must be created manually and synced using something like SSSD. Mappings from users to their group membership are also required in which LDAP or SSSD can be used. If this is a requirement, we recommend that you use LDAP to authenticate users, and use Amazon EMR with Kerberos to automate the OS accounts syncing. Or, you can also enable Hadoop to do user-to-group mappings via LDAP. See Hadoop Groups Mappings in the Apache Hadoop documentation for built-in Hadoop implementations for these mappings. For more information, refer to Appendix C: Sample LDAP Configurations.

The following table lists Amazon EMR supported applications and instructions on how to enable LDAP, if supported.

Table 1: Amazon EMR Supported Applications and LDAP Support

Application	Supports LDAP?	Notes
Apache Hive	Yes	HiveServer2 can be used to authenticate against LDAP. Details on HiveServer2 setup are located on the Apache Hive Wiki. When LDAP is configured, Java Database Connectivity (JDBC) and Beeline connections require a user login and password. **Note:** We recommend that you use LDAPS to ensure that unencrypted credentials are not shared.
Presto	Yes	See LDAP Authentication in Presto documentation for details on the different configuration parameters.

Application	Supports LDAP?	Notes
Apache Spark	No	Use Hue or Apache Zeppelin to authenticate the user and then submit the job to Spark. Or use Kerberos as an authentication mechanism when submitting applications.
Apache Hue	Yes	See Configure Hue For LDAP users in the *Amazon EMR Release Guide* for setup instructions, and Using LDAP via AWS Directory Service to Access and Administer Your Hadoop Environment on the *AWS Big Data Blog* for a step-by-step guide.
Apache Zeppelin	Yes	Apache Zeppelin uses Apache Shiro to configure authentication. For steps on enabling Shiro, see Shiro Authentication for Apache Zeppelin.
Apache HBase	No	Use Kerberos as an authentication mechanism.
JupyterHub	Yes	See Using LDAP Authentication in the *Amazon EMR Release Guide* for setup instructions.

Kerberos

Kerberos is the most secure authentication and authorization mechanism available on Amazon EMR. Kerberos works by having users provide their credentials and obtain a ticket to prove identity from a central Authentication Server. That ticket can then be used to grant access to resources within your cluster. For Kerberos to be used effectively for authentication, configure your Amazon EMR clusters to create a one-way trust between the Kerberos server running on the Amazon EMR master node and the Kerberos server that is on-premises. An example flow is provided in EMR Kerberos Flow for directly interacting with HiveServer2.

Most applications within Amazon EMR support Kerberos as an authentication method. For a complete list, see Supported Applications in the *Amazon EMR Management Guide*.

Authentication Between Applications

Similar to users, applications and service accounts require authentication when interacting with each other. There are several ways this happens within Amazon EMR (see *Figure 8*).

Kerberos

Kerberos is the recommended method for application-to-application authentication. When using Kerberos authentication, applications authenticate themselves with a Key Distribution Center (KDC), and authenticate other applications connecting to it. There are three options when using Kerberos for EMR clusters: Cluster-Dedicated KDC, Cross-Realm Trust and External KDC. See Kerberos Architecture Options for details about each option. For more information on Kerberos enabled workflows on Amazon EMR, see Appendix B: EMR Kerberos Workflow.

Presto

The Amazon EMR version of Presto allows nodes within a cluster to authenticate using LDAPS. To set up node-to-node authentication, set up Presto using LDAPS. See Using SSL/TLS and Configuring LDAPS with Presto on Amazon EMR in the *Amazon EMR Release Guide* for details.

SSL/TLS Between Nodes

See Encryption for Data in Transit for how to enable SSL/TLS between nodes through Amazon EMR security configuration. Using SSL certificates to authenticate between nodes can provide protection against a rogue node from joining a cluster.

Authorization

Authorization is the act of allowing or denying an identity to perform an action. Using authorization to determine what an identity can do first requires that the identity has validated who they are. This section provides the various mechanisms available that can limit what an identity can do.

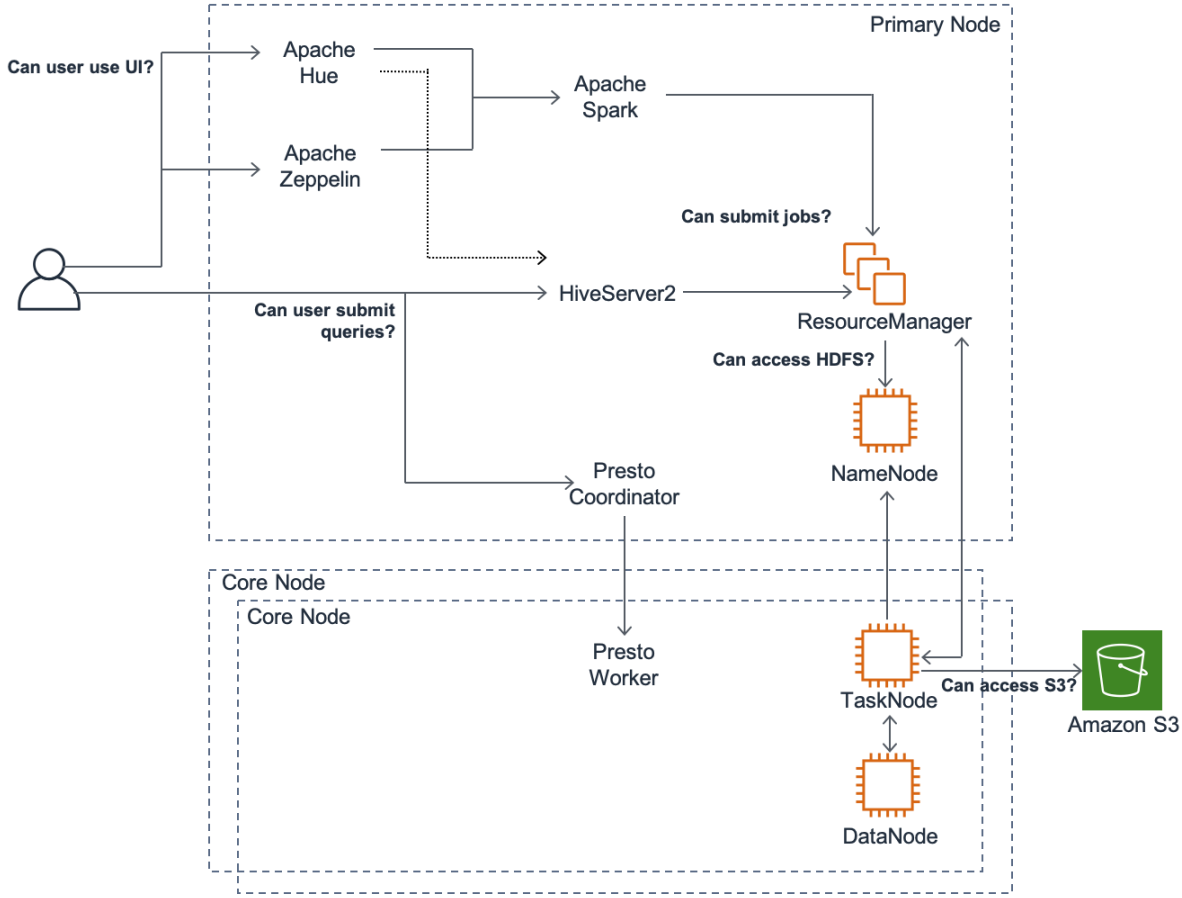


Figure 9: Example of authorization flow

Application Authorization

For Apache Hadoop, there are several possible solutions for authorization, like having individual applications control what users can do, or having central processes or applications that can manage policies across several applications.

Apache Knox

Apache Knox provides perimeter security for your Hadoop resources. Users connect to Knox and can be authenticated and authorized through a Knox Gateway. The gateway then forwards the user's traffic to Hadoop resources without having to directly connect to them. Knox can be configured to allow groups to dictate the resources a user may access, such as Apache HBase UIs.

For more information, see Authentication in the *Apache Knox Gateway Users Guide* and How to Implement Perimeter Security with Amazon EMR using Apache Knox.

Hue, Zeppelin, Hive (LDAP)

For applications that integrate with LDAP, such as Apache Hue, Apache Zeppelin, and Apache Hive, these applications can be set up to only allow users that belong to certain LDAP groups. See the following links for helpful resources:

- Authenticate Hue Users with LDAP
- HiveServer2 Authentication/Security Configuration
- Configure Zeppelin for Authentication: LDAP and Active Directory

Storage Level Authorization (Amazon S3)

If you have Amazon EMR clusters with multiple users who need different levels of access to data in Amazon S3, you have multiple options. Implementation depends on the use case and each approach has trade-offs. You can segregate your EMR clusters, have EMR File System (EMRFS) assume different AWS IAM roles, or use AWS Lake Formation integration with Amazon EMR.

Separation of EMR Clusters for Different Permissions

You can segregate permissions among different groups of users to create individual EMR clusters. As a default configuration for EMR accessing Amazon S3, EMRFS assumes the permissions policies attached to the Service Role for Cluster EC2 Instances (EC2 Instance Profile) specified when the cluster was created. In this case, the same service role for cluster EC2 instances is used regardless of the user or group using the application or the location of the data in Amazon S3. For example, you can create an EMR cluster that is dedicated to ML users that have access to appropriate data for analysis. You can then create a separate cluster for end users that only has access to the output of the analyses that the ML users created.

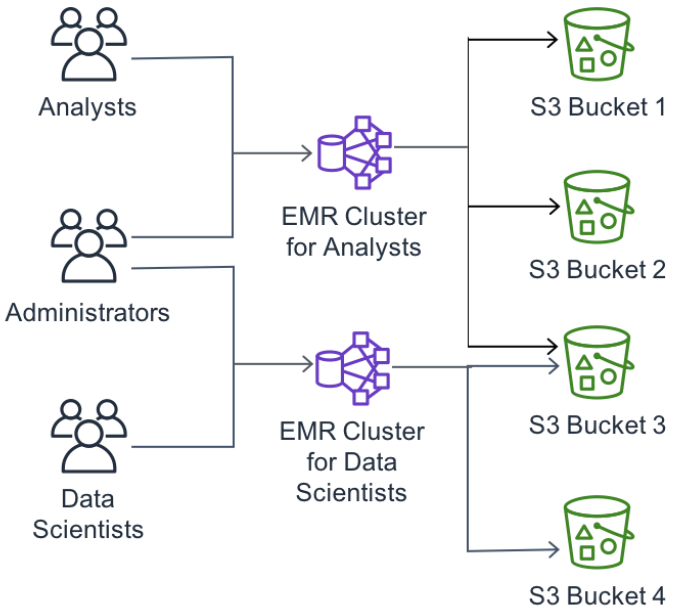


Figure 10: Separation of EMR clusters for different permissions

EMR File System (EMRFS) Authorization

EMRFS Authorization allows a single cluster to be used by multiple groups of users while providing different S3 permissions for different users. Depending on the user, that user's group, or the S3 location, EMRFS can assume a different role and interact with S3 based on the permissions of that role.

A role mapping is required between a user, group, or S3 location, and the role that EMRFS assumes.

For example, consider two users, *user A* and *user B*, and two AWS IAM Roles, *analyst* and *data_scientist*. An EMRFS mapping may specify that when *user A* is accessing S3, the ERMFS assumes the *analyst* role. When *user B* accesses S3, the EMRFS assumes the *data_scientist* role.

If using Kerberos for Hadoop YARN, then it is required that the user's accounts are registered on each of the nodes on Hadoop and that there is a defined way to map users to groups. You can create this mapping manually through bootstrap actions, or by having a mechanism to access users from an on-premises Active Directory. To automate this process, you can use SSSD to read identity information, including user-to-group mappings. AWS automates the setup of a one-way trust using Kerberos with EMR. See Use Kerberos Authentication for more details.

You can update rules while the cluster is running by updating `emrfs-site.xml` on all nodes on the cluster. The EMRFS will detect and apply the new mappings.

> **Note**: IAM roles for EMRFS provide application-level isolation between users of the application. This configuration does not provide host level isolation between users on the host. Any user with access to the cluster can bypass the isolation to assume any of the roles. This method is not recommended if users can SSH into your cluster to run arbitrary code, such as running Apache Spark, Scala, or PySpark jobs.

AWS Lake Formation Integration with Amazon EMR

AWS Lake Formation is a service that makes it easier to set up a secure data lake in days. By integrating Amazon EMR with AWS Lake Formation, you can achieve fine-grained column-level access to databases and tables in the AWS Glue Data Catalog, as well as enable federated single sign-on to EMR notebooks or Apache Zeppelin from your enterprise identity system that is compatible with Security Assertion Markup Language (SAML) 2.0.

Simply register your Amazon Simple Storage Service (Amazon S3) buckets and paths with AWS Lake Formation and create data catalog resources using the AWS Lake Formation console, the API, the AWS Command Line Interface (AWS CLI), or using AWS Glue Crawlers. See the AWS Lake Formation Developer Guide for more details on how to set up the data lake.

Once you set up the data lake, you can grant permissions to access the data using the AWS Lake Formation permissions model. This permissions model is similar to DBMS-style `GRANT/REVOKE` commands, such as `Grant SELECT` on `tableName` to `userName`. You can also limit the access to specific columns with an optional inclusion or exclusion list. The inclusion list specifies the columns that are granted access, and the exclusion list specifies the columns that are not granted access. In the absence of an inclusion or exclusion list, access is granted to all columns in a table. See AWS Lake Formation Access Control Overview on how to set up these permissions.

If you are already using AWS Glue identity-based polices or AWS Glue resource policies for access control, you can manage these policies in the AWS Lake Formation Permission model by identifying existing users and roles and setting up equivalent Lake Formation permissions. See Upgrading AWS Glue Data Permissions to AWS Lake Formation Model for more details.

Currently, you can use either Amazon EMR Notebooks or Apache Zeppelin to read this S3 data using Spark SQL with Apache Livy. The user first needs to authenticate using one of the third-party SAML providers (Microsoft Active Directory Federation Services [AD FS], Auth0, and Okta). See Supported Third-party Providers for SAML on how to set up trust relationships between SAML providers and AWS Identity and Access Management (IAM).

See IAM Roles for Lake Formation for more details on IAM Roles needed for this integration and Launch an Amazon EMR cluster with Lake Formation on how to launch Amazon EMR cluster.

Amazon EMR enables fine-grained access control with Lake Formation by using the following components:

- Proxy agent: The proxy agent is based on Apache Knox. It receives SAML-authenticated requests from users and translates SAML claims to temporary credentials. It also stores the temporary credentials in the secret agent which runs on the master node.
- Secret agent: The secret agent securely stores secrets and distributes secrets to other EMR components or applications. The secrets can include temporary user credentials, encryption keys, or Kerberos tickets. The secret agent runs on every node in the cluster and uses Lake Formation and AWS Glue APIs to retrieve temporary credentials and AWS Glue Data Catalog metadata.
- Record server: The record server receives requests for accessing data. It then authorizes requests based on temporary credentials and table access control policies distributed by the secret agent. The record server reads data from Amazon S3 and returns column-level data that the user is authorized to access and runs on master node.

See Architecture of SAML-Enabled Single Sign-On and Fine-Grained Access Control for an illustration and details of fine-grained access control.

Metadata Authorization

Hive, Presto, and Spark SQL all leverage a table metastore, also known as a data catalog, that provides metadata information about the tables that are available to be queried. You can use metadata to restrict users' access to specific datasets.

Apache Ranger

Apache Ranger provides a set of plugins that you can install and configure on supported applications, such as Hive, HDFS, and YARN. The plugins are invoked during user actions and allow or deny actions based on policies you provide to the Ranger server. Ranger can set policies on databases and tables within Hive to allow users and groups to access a subset of Hive tables and databases. Ranger also provides the ability to do column-level authorization, column masking, and row-level filtering. The policies are managed on the Ranger web console, and they can interact with your LDAP server.

For details on how to achieve this on Amazon EMR, see Implementing Authorization and Auditing using Apache Ranger on Amazon EMR on the *AWS Big Data Blog*.

Hive SQL Authorization

Hive SQL authorization provides another mechanism for you to control which users have access to tables and databases. For more information, see SQL Standard Based Hive Authorization on the Apache Hive Wiki.

AWS Data Catalog Authorization

The AWS Glue Data Catalog allows you to restrict access to databases or tables based on the IAM principals & IAM policies. See Authentication and Access Control for AWS Glue for more details.

AWS Lake Formation Metadata Authorization

AWS Lake Formation provides centralized metadata permissions management for data catalog administrators. These permissions enable principals to create, read, update, and delete metadata databases and tables in the data catalog. AWS Lake Formation also centralizes and enables cross-account access control to AWS Glue Data Catalog metadata and underlying data making it a valuable tool for enterprises employing a secure, multi-account strategy. For more information on this feature, see Cross-Account Access.

The Lake Formation model is implemented as a DBMS-style `GRANT/REVOKE` command set, whereas IAM provides the IAM policy as a tool to define permissions on resources. For data catalog requests to succeed, both IAM and Lake Formation must permit the access.

The recommended approach to securing Data Catalog resources is to enable coarse-grained permissions on access to the underlying AWS Glue API actions and use Lake Formation to govern fine-grained access control to the databases, tables, and columns in the Data Catalog.

IAM Coarse-Grained Metadata Permissions

Lake Formation works well with clearly defined roles and personas that align with the access patterns of your data lake users.

Typically, Data Catalog administrator roles hold the following AWS managed IAM policies:

- AWSLakeFormationDataAdmin
- AWSGlueConsoleFullAccess (optional)
- CloudWatchLogsReadOnlyAccess (optional)
- AmazonAthenaFullAccess (optional)

For more information on the associated IAM policies for common Lake Formation personas, such as the Data Engineer, Data Analyst and Workflow Roles, see Lake Formation Personas and IAM Permissions Reference.

Lake Formation Fine-Grained Metadata Permissions

With IAM providing access to the underlying AWS Glue and Lake Formation APIs, Lake Formation can be used to centralize and simplify Data Catalog permissions. Access to Data Catalog resources can be controlled down to the column level using either the Lake Formation console or the Lake Formation CLI.

The following table summarizes the available Lake Formation permissions on Data Catalog resources. Items with an asterisk (*) affect underlying data access but also limit search capabilities in the Lake Formation console.

Table 2: Data Catalog Permission Summary

Catalog	Database	Table	Column
CREATE_DATABASE	CREATE_TABLE	ALTER	SELECT*
	ALTER	DROP	
	DROP	SELECT*	
		INSERT*	
		DELETE*	

Administrators can grant permissions to create, alter, or drop databases or tables as well as permissions that affect underlying data consumption such as permissions on select, insert, or delete operations at the table or column level. These operations can be done either in the Lake Formation console or via the Lake Formation CLI. The example command below grants the user datalake_user1 permission to create tables in the retail database:

```
aws lakeformation grant-permissions –principal
DataLakePrincipalIdentifier=arn:aws:iam::111122223333:user/datalake_user1
--permissions "CREATE_TABLE" --resource '{ "Database": {"Name":"retail"}}`
```

Explicit Permissions

Lake Formation permission grants have the form:

Grant *permissions* to *principal* on *resource* [with grant option].

With the grant option, you can allow the grantee to grant permissions to other principals. *Table 2* highlights the explicit permissions that can be granted by the resource type. For example, when `CREATE_TABLE` is granted to a principal on a database, the principal is allowed to create tables in that database.

Permissions marked with an asterisk (*) in Table 1 are granted on Data Catalog resources but also apply to the underlying data. For example, the DROP permission on a metadata table enables the principal to drop the referenced table from the Data Catalog. However, the `DELETE` permission, when granted on the same table, gives explicit permissions to the principal for deleting the table's underlying data in Amazon S3. As such, `SELECT`, `INSERT`, and `DELETE` permissions effect both Data Catalog and data access permissions. For more information see Lake Formation Access Control Overview.

Implicit Permissions

It's important to keep in mind the following implicit metadata permissions that are granted by Lake Formation:

- Database creators have all database permissions on databases that they create, and can grant others permission to create tables in the database.

- Database creators do not gain implicit permissions on tables that others create in the database.
- Table creators have full permissions on tables they create, can grant permissions on tables they create and can view databases that contain tables that they create.

Be sure to remove any IAM users or roles in the IAMAllowedPrincipals group from the AWS Lake Formation permissions if any exist. The IAMAllowedPrincipals is automatically created and includes all IAM users and roles that are permitted access to your Data Catalog resources by IAM. This mechanism is provided for existing customers who are transitioning into AWS Lake Formation. Once the on-boarding to AWS Lake Formation is complete, permissions to the IAMAllowedPrincipals group should be revoked leaving only AWS Lake Formation in charge of access control as is recommended. For more information, see Changing the Default Security Settings For Your Data Lake.

For more information regarding implicit permissions, see Implicit Lake Formation Permissions.

Amazon EMR and AWS Lake Formation Metadata Authorization

General Operations and Administrative Flow

Lake Formation simplifies and centralizes metadata authorization for Amazon EMR. The integration of AWS Lake Formation with Amazon EMR makes use of three required IAM roles for coarse-grained permissions and relies on Lake Formation as the single defining source for fine-grained metadata permissions. (See Overview of the IAM Roles for Lake Formation for more details.) This also includes the ability to grant other users permissions. Administration of these permissions can be done through the Lake Formation console or the Lake Formation CLI for automated operations.

Outside of IAM, there are a few configuration requirements that must be met for Amazon EMR clusters to integrate with AWS Lake Formation:

- Enable Kerberos authentication using the cluster-dedicated KDC
- Configure your Amazon EC2 security group or Amazon VPC network access control list (ACL) to allow access to the proxy agent (port 8442) from your user desktops

In addition, it's recommended that you enable encryption in transit and at rest. Optionally, you can create a custom TLS key pair to further secure communications with the proxy agent. For a full description of these requirements and setup information, see Configure EMR Security Features.

With the required IAM roles in place, data catalog administration can be controlled entirely via Lake Formation and enables secure, fine-grained access to data for federated EMR users. Lake Formation also enables column-level permissions to AWS Glue Data Catalog resources. Access to other AWS services (i.e. those not managed by Lake Formation, such as DynamoDB) is controlled via the IAM Role for AWS Services described below. Apache Spark job submission can be done through EMR notebooks, Apache Zeppelin, or Apache Livy. See Supported Applications and Features for applications and features that support the integration of AWS Lake Formation with Amazon EMR. See also Implementing SAML AuthN for Amazon EMR Using Okta and Column-Level AuthZ with AWS Lake Formation for a blog post article

that demonstrates applying column-level authorization with AWS Lake Formation while authenticating with an independent identity provider (IdP).

IAM Role for AWS Services (Access to Non-Lake Formation Services)
This role defines the permissions that the EMR cluster has when accessing services not managed by AWS Lake Formation Services. If the jobs on your cluster require access to any other AWS services, such as DynamoDB, this role must contain the policies required to access those services. It's important to **not** include permissions for the following operations in this role to keep the separation between IAM and Lake Formation clear:

- AWS Glue API operations
- AWS Lake Formation API operations
- AWS STS AssumeRole operations
- Amazon S3 access to buckets managed by AWS Glue

> **Note:** These resources should be accessed through Lake Formation using Spark SQL only.

Amazon EC2 Instance Profile
The EC2 instance profile is a special type of service role that defines permissions for EMR clusters to interact with Lake Formation and other AWS services. The following policy must be included referencing both the IAM role for Lake Formation as well as the IAM Role for AWS Services described previously. Note the inclusion of Amazon S3 bucket/prefix access to the metadata XML file generated by your IdP. For more information, see Configure Trust Relationship Between your IdP and Lake Formation.

```
{
    "Version": "2012-10-17",
    "Statement": [
      {
         "Effect": "Allow",
         "Action": [
              "s3:GetObject",
              "s3:ListBucket"
         ],
         "Resource": "arn:aws:s3:::BUCKET_AND_PREFIX_FOR_SAML_XML_METADATA_FILE"
    },
        {
         "Effect": "Allow",
         "Action": "iam:PassRole",
         "Resource": "arn:aws:iam::account-
id:role/IAM_SAML_Role_For_Lake_Formation"
        },
```

```
        {
         "Effect": "Allow",
         "Action": "sts:AssumeRole",
         "Resource": "arn:aws:iam::account-id:role/IAM_Role_for_AWS_Services"
        },
        {
         "Effect": "Allow",
         "Action": "lakeformation:GetTemporaryUserCredentialsWithSAML",
         "Resource": "*"
        },
        {
         "Effect": "Allow",
         "Action": "iam:GetRole",
         "Resource": [
            "arn:aws:iam::account-id:role/IAM_SAML_Role_For_Lake_Formation",
            "arn:aws:iam::account-id:role/IAM_Role_for_AWS_Services"
            ]
        }
    ]
}
```

Encryption

You can implement data encryption to help protect data in transit, and data at rest in Amazon S3 and in cluster instance storage. You can also use a custom Amazon Linux AMI to encrypt the Amazon Elastic Block Store (Amazon EBS) root device volume of cluster instances.

The following diagram shows the different data encryption options that are available with security configurations.

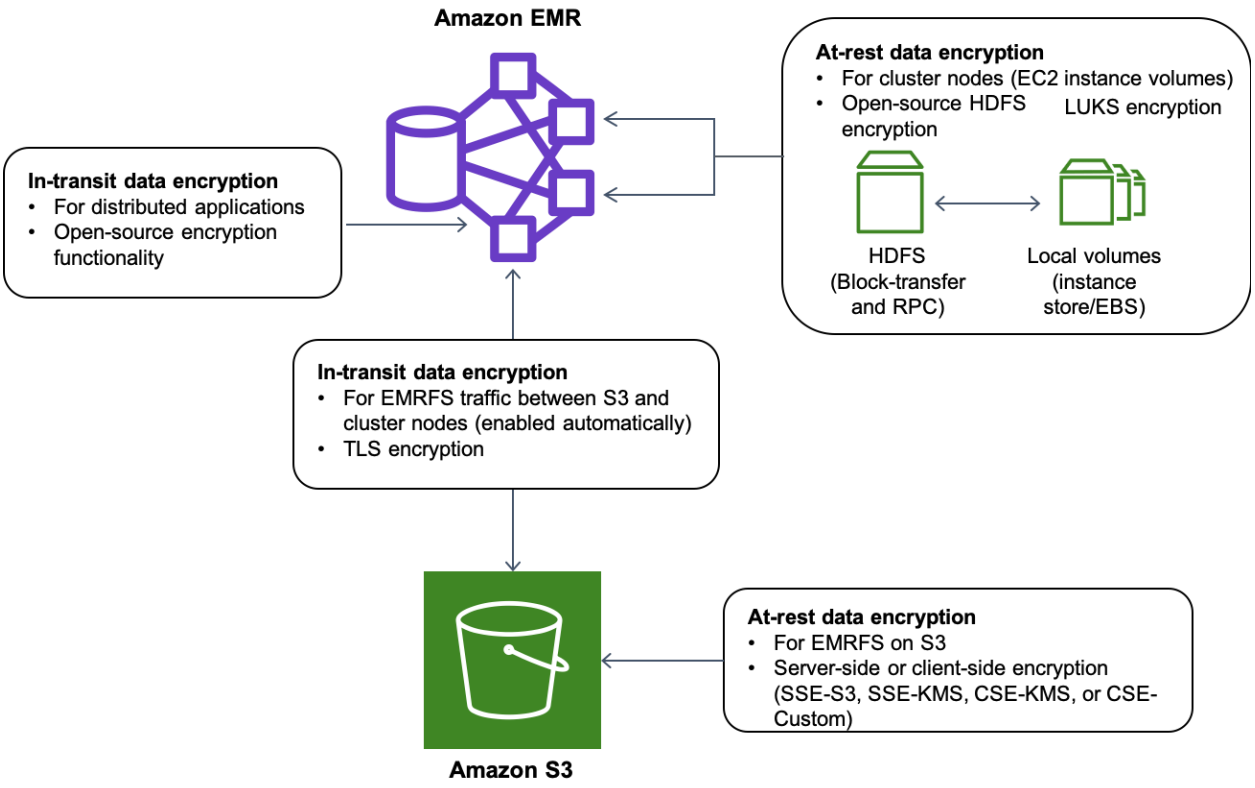


Figure 11: Encryption options

For an up-to-date list of encryption options available on Amazon EMR, see Encryption Options.

Encryption for Data in Transit

Amazon EMR security configurations enable you to choose a method for encrypting data in transit using Transport Layer Security (TLS). You can do either of the following:

- Manually create PEM certificates, zip them in a file, and reference them from Amazon S3
- Implement a certificate custom provider in Java and specify the S3 path to the JAR

See Encrypt Data in Transit and At Rest in the *Amazon EMR Management Guide* for details on how to use EMR security configuration to configure encryption in transit and which applications are supported. We highly recommend that you encrypt all interaction with your cluster, including all UI-based applications and endpoints.

Encryption for Data at Rest

You have multiple options to encrypt data at rest in your EMR clusters. By default, Amazon EMR uses the EMR File System (EMRFS) to read from and write data to Amazon S3. To encrypt data in Amazon S3, you can specify one of the following options:

- **SSE-S3:** Amazon S3 manages the encryption keys for you.
- **SSE-KMS:** You use an AWS Key Management Service (AWS KMS) customer master key (CMK) to encrypt your data server-side on Amazon S3. Be sure to use policies that allow access by Amazon EMR.
- **CSE-KMS/CSE-C:** Amazon S3 encryption and decryption takes place client-side on your Amazon EMR cluster. You can use keys provided by AWS KMS (CSE-KMS) or use a custom Java class that provides the master key (CSE-C).

EMR security configuration also supports encryption overrides per bucket, i.e. using separate KMS keys to encrypt objects in specific S3 buckets. See Encrypt Data in Transit and At Rest in the *Amazon EMR Management Guide* for details on how to use EMR security configuration.

Root Volume

Beginning with Amazon EMR version 5.24.0, you can use the Amazon EMR security configuration option to encrypt the EBS root volume. For Amazon EMR versions prior to 5.24.0, you must use a Custom AMI to encrypt the root volume. See Creating a Custom AMI with an Encrypted Amazon EBS Root Device Volume in the *Amazon EMR Management Guide* for details.

EBS Volumes

There are two mechanisms that allow you to encrypt data on non-root volumes, which typically store HDFS and other application data: HDFS Transparent Encryption for application-managed encryption and LUKS encryption for OS-managed encryption. For documentation on both approaches, see At-rest Encryption for Local Disks in the *Amazon EMR Management Guide*.

For Amazon EMR version 5.24.0 and later, you can natively encrypt EBS volumes attached to an EMR cluster by using the Amazon EMR security configuration option. EBS encryption provides the following benefits:

- Native End-to-End Encryption: Data on EBS volumes including intermediate data, I/O between the EC2 instances and EBS volumes, and EBS snapshot are encrypted.
- Root Volumes Encryption: Root volumes can be encrypted without the need to create custom Amazon Linux AMIs.
- Transparent Encryption: EBS encryption is transparent to any applications running on EMR and does not require modifications.
- Simplified Encryption: With EBS encryption, you can check encryptions status from the Volumes page in the EC2 console or through an EC2 API call.

Perimeter Security

Establishing perimeter security is foundational to Amazon EMR Security. This process involves limiting access to the EMR cluster itself and using an “edge” node (gateway) as a host. Users and systems can

use this edge node directly and then have all other actions issued from this gateway into the cluster itself.

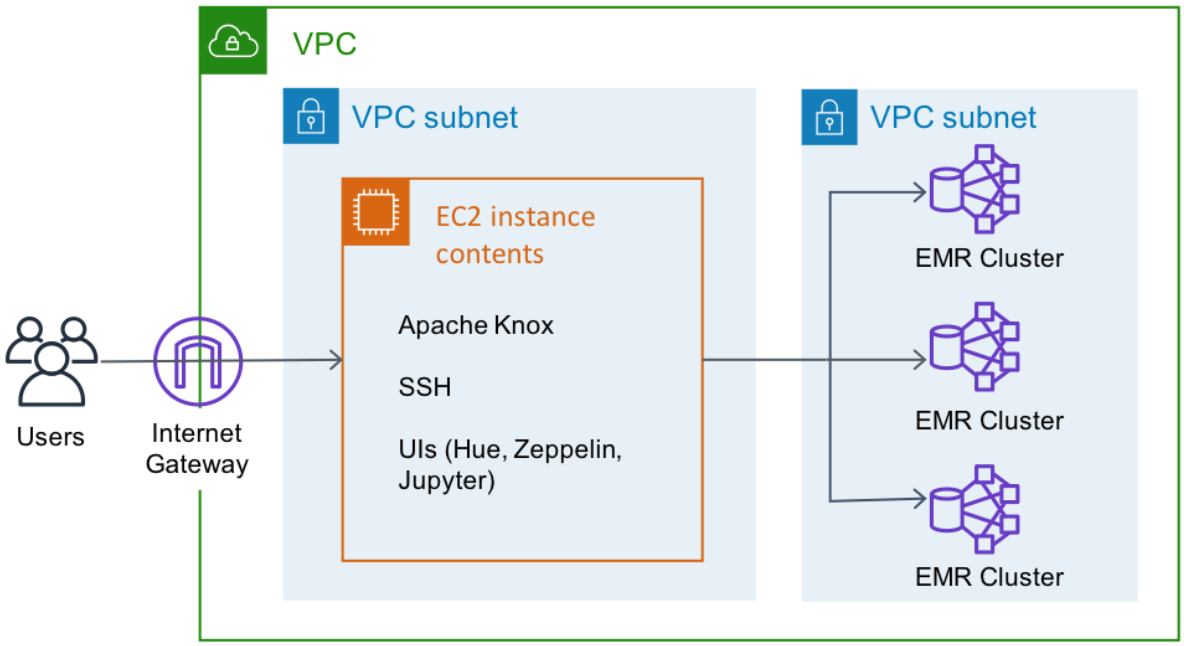


Figure 12: Perimeter security

Apache Knox

Apache Knox is an Application Gateway that provides a single endpoint to interact with multiple applications and multiple clusters, as well as an authentication layer to the applications and clusters that supports SAML, LDAP, and other authentication methods. Knox is useful for several use cases:

- If you require HTTPS between the client and Apache Knox: Apache Knox supports TLS so clients can connect to Knox over a secure channel, regardless if the application on the cluster supports TLS or not.
- If you require an authentication layer: Apache Knox provides an authentication layer that supports several types of authentication mechanisms that can be used for instances where the cluster application does not support it. For example, Knox supports single sign-on (SSO) which can be used as Hadoop Resource Manager does not support it.
- Apache Knox records actions that are executed per user request or those that are produced by Knox internal events.

However, Apache Knox does have some limitations, and it does not provide fine-grained access control.

Network Security

This document does not provide information on how to properly set up a Virtual Private Cloud (VPC), subnets, security groups, and network access control lists (ACLs). For information on setting up VPCs, see Control Network Traffic with Security Groups.

Best Practices for Network Security

- If your cluster is intended only for batch processing and a user or another application will not need to access it, launch the cluster in a private subnet with an Amazon S3 endpoint and any other required endpoints. See Configure Networking for more information.
- If your cluster is in a private subnet, and it requires access to an AWS service that does not have a VPC endpoint or requires access to the internet, create a NAT instance. See Scenario 2: VPC with Public and Private Subnets (NAT) for more information.
- If using NAT instance/gateway or internet gateway is not an option, you can configure a proxy server to connect to the internet. Proxy servers provide varying levels of functionality, security, and privacy depending on your use case, needs, or company policy. Amazon EMR cluster provides options to configure applications to route the network traffic via a proxy server. You can use the `additional-info` parameter when configuring the cluster in these scenarios. See create-cluster in the *AWS CLI Command Reference* for more information on the `additional-info` parameter.
- If you are using a private subnet and creating an S3 endpoint, use a restrictive S3 policy for the endpoint. The minimum policy requirement is documented at Amazon EMR 2.x and 3.x AMI Versions.
- If you require your cluster to be in a private subnet, but need interactive access to the cluster, create a bastion host (also known as an edge node) that sits in a public subnet and connect to your Amazon EMR cluster from it. This setup allows you to enforce security policies on the edge node, but not on the Amazon EMR cluster. See Securely Access Web Interfaces on Amazon EMR Launched in a Private Subnet on the *AWS Big Data Blog* for a step-by-step guide.
- If you can forgo SSH access to your non-interactive cluster, it can simplify security requirements on the Amazon EMR cluster.
- If you don't want the communication between your VPC and cluster to go over the internet, connect directly from your VPC to Amazon EMR cluster using an interface VPC endpoint. When you use an interface VPC endpoint, communication between your VPC and Amazon EMR is conducted entirely within the AWS network. See Connect to Amazon EMR Using an Interface VPC Endpoint for more information.

- If you do not want unintentional public access to applications running on the cluster, enable *Block public access*. Block public access can be configured and enabled for each AWS Region for your account. The Block public access feature is supported only for Amazon EMR clusters running in public subnets and is not applicable to Amazon EMR clusters running in private subnets. Block public access prevents a cluster from launching when any security group associated with the cluster has a rule that allows inbound traffic from IPv4 0.0.0.0/0 or IPv6 ::/0 (public access) on a port, unless the port has been specified as an exception (Port 22 is an exception by default). You can configure exceptions to allow public access on a port or range of ports. You can also disable block public access but it's recommended to enable it. See Using Amazon EMR Block Public Access for more information

Auditing

The ability to audit compute environments and understand where the data in the cluster is coming from and how it's being used is a key requirement for many customers. There are a variety of ways that you can support this requirement within EMR.

Amazon S3

Amazon EMR can be configured to push all application, daemon, and provisioning logs to Amazon S3 by enabling logging. See Configure Cluster Logging and Debugging for more information.

Apache Ranger

Apache Ranger plugins provide the ability to push auditing data into Apache Solr for future analysis. Apache Ranger also pushes auditing information within the Ranger UI.

AWS CloudTrail

AWS CloudTrail documents every API call made to AWS services and contains information such as the callers AWS access key, source IP, agent used make the call, and so on. When using Amazon EMR, there are several groups of API calls that are stored. The first group is at the EMR service level, where APIs are called for cluster management, such as creating a cluster, terminating clusters, and running steps. Secondly, when a running cluster is attempting to access a resource, such as S3, Glue Data Catalog, and KMS, the credentials that the cluster used are stored in AWS CloudTrail. See Logging Amazon EMR API Calls in AWS CloudTrail for more details on how to enable and use CloudTrail.

To help meet compliance requirements, CloudTrail logs can be exported to Amazon S3, and then queried by Amazon Athena to be able to detect unauthorized or suspicious activity. See Querying AWS CloudTrail Logs for more information on how to query from Amazon Athena.

Amazon CloudWatch Logs

Rather than using the EMR log pusher to push logs to S3, Amazon CloudWatch Log agent can be used to push logs to CloudWatch Logs. This approach has the benefit of making the logs searchable, and being able to stream these logs to another system, such as Amazon Elasticsearch Service, or use AWS Lambda to push the logs to other systems, like Splunk.

To use this approach, you must install and configure the Amazon CloudWatch Agent, and configure it to watch the Hadoop log directories. It then starts to push logs to Amazon CloudWatch Logs, where you can set up log retention and export targets based on your auditing requirements.

Apache Knox

If you are using Apache Knox as a gateway for perimeter security or proxying, you can also configure it to log all actions a user does on all resources that it interacts with.

Software Patching

Amazon EMR provides new releases on a regular basis, adding new features, new applications, and general updates. We recommend that you use the latest release to launch your cluster whenever possible.

OS Level Patching

When an EC2 instance starts, it applies all OS non-kernel patches. You can view installed patches by running `yum updateinfo list security installed` from the AWS Command Line Interface (AWS CLI).

To install other OS level patches, we recommend three approaches:

- Upgrade to a newer AMI: The latest patches, including Linux kernel patches, are applied to newer AMIs. See the Release Notes for which versions have upgraded applications and check the application release notes to see if there is a fix for your issue.
- If you require more flexibility over which patches are applied, use custom AMIs. This approach allows you to apply new patches to the AMI and deploy them by restarting your clusters.
- Contact AWS Support and provide the issue and patches that you need. Support may be able to provide workarounds.

If you have a long running cluster, you can use EC2 Patch Manager. This approach requires that your instance has SSM up and running. EMR cluster patching requires that you apply patches to a cluster in a maintenance window so that nodes can be restarted in bulk, especially the primary node. Or, you can apply patches to one node at a time to ensure that HDFS and jobs are not interrupted. Make sure to contact AWS Support before applying any patches. If instances must be rebooted during the patch

process, make sure to turn on termination protection before patching. See Using Termination Protection for details.

For more information on how to push updates using EC2 System Manager, see the *AWS Big Data Blog* post Create Custom AMIs and Push Updates to a Running Amazon EMR Cluster Using Amazon EC2 Systems Manager.

Application Level Patching

There are a couple options to patch applications within Amazon EMR:

- Upgrade to a newer AMI: Newer AMIs include recent patches. View the Release Notes for which versions have upgraded applications and check the applications release notes to see if there is a fix for your issue.
- Contact AWS Support and provide the issue and patches that you need. Depending on severity, Support may provide other workarounds or in extreme cases, may be able to create a bootstrap action that provides the patch.

> **Note:** We recommend that you do not apply and install patches on your cluster unless you have contacted AWS Support and the EMR service team. In certain software upgrade scenarios, you may need to assume operational burden of maintaining the patches.

Software Upgrades

General recommendations for upgrading Amazon EMR AMIs are:

- Read the Amazon EMR Release Notes to see version changes for applications, such as Hive, HBase, and so on.
- Read the release notes of the open source software packages, such as Hive, HBase, and so on, and check for any known issues that could impact existing workflows.
- Use a test environment to test the upgraded software. This approach is important to isolate testing workloads from production workloads. For example, some version upgrades of Hive update your Hive Metastore schemas which may cause compatibility issues. If someone starts and uses Hive with the newer version, it could impact production.
- Test performance and data quality between the old and new versions. When moving versions in Hive, compatibility issues or bugs may occur that can impact data quality and performance.

Common Customer Use Cases

Batch Processing EMR Clusters in a Private Subnet

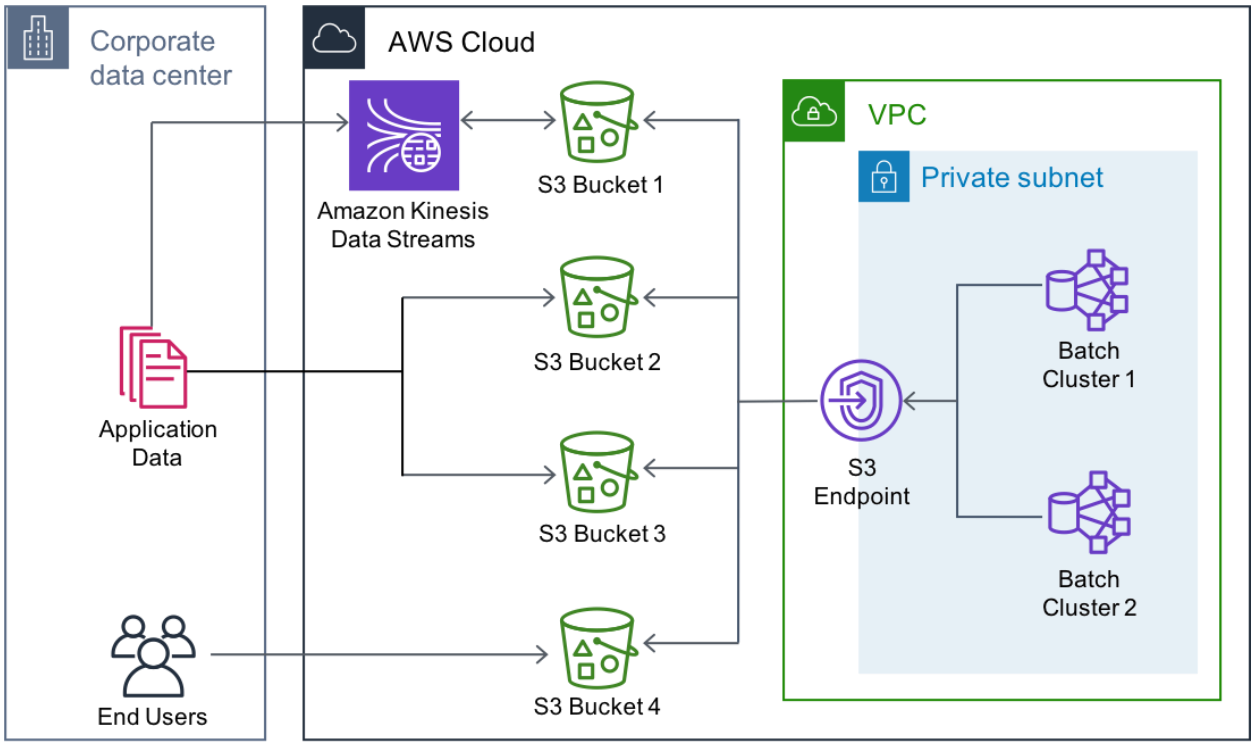


Figure 13: Batch processing EMR clusters in a private subnet

Use Case

- Secured EMR clusters batch processing data in ETL fashion.
- Users do not need to directly access the clusters.

When To Use

- Having an ETL-only use case may allow for reduced security requirements compared to clusters in which users interactively access.
- This design works well when Amazon EMR clusters are being used as transient clusters.

Implementation

1. Create a VPC and private subnet with an S3 endpoint.
2. Disable SSH to the EMR clusters.
3. If extra security is required, complete these steps through security configuration:
    a. Configure Kerberos without a one-way trust to enable application-to-application security.

b. Enable encryption on data in transit and data at rest.

4. Implement S3 bucket policies to read-only access requests from the VPC and write access from end users and applications writing to S3.
5. Secure EMR API calls using IAM policies to restrict access to clusters that require it.
6. Submit work to the EMR cluster by one of the following ways:
   - Use EMR Steps API to submit work.
   - Install Oozie, Airflow, or a similar application on the master node and have it orchestrate your workloads.

Using a Bastion Host/Edge Node

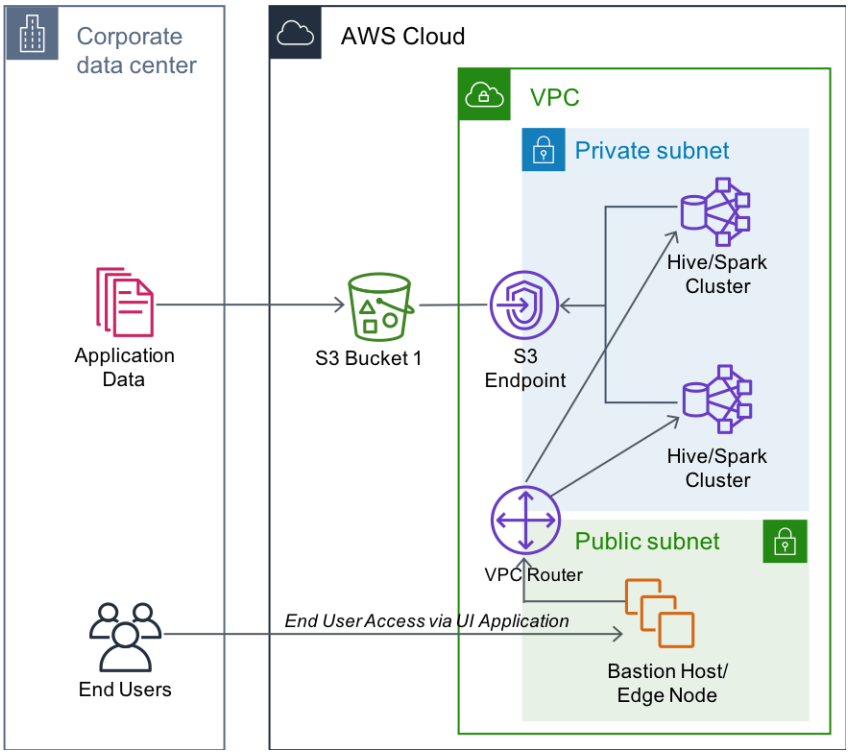


Figure 14: Using a bastion host/edge node

Use Case

- Securing an EMR cluster is not necessary because it's running in a private subnet with highly controlled access controls to it.
- You can control which AMI you want to use on the bastion host. For example, if users can only access an internally managed Red Hat Linux, then bastion host can run that AMI, while EMR clusters continue to use Amazon Linux.

- Once a user is authenticated on a bastion host, then they should have access to all the resources on the EMR cluster.

When to Use

- You want to control access to the EMR cluster and have all users interact with EMR through the bastion host/edge node.
- You want to segregate job submission and EMR processes running on the master and data nodes.

Implementation

1. Create EMR clusters within a private subnet with VPC endpoints that allow traffic to required AWS services, like Amazon S3, AWS KMS, and Amazon DynamoDB.
2. Create a bastion host on an EC2 instance in a public subnet. Traffic from the public subnet must be able to access the clusters in the private subnet.
3. Do not allow SSH access to EMR clusters.
4. Access the clusters through a UI, like Hue, Zeppelin, or Jupyter, or provide SSH access to the bastion host/edge node.
5. If SSH is allowed, ensure that the bastion host/edge node uses SSSD or another mechanism to allow users to log in with their on-premises credentials.
6. Use an elastic load balancer (ELB) that routes traffic to clusters for load balancing and disaster recovery without needing to change configurations on the bastion host.

Separate EMR Clusters for Different Use Cases

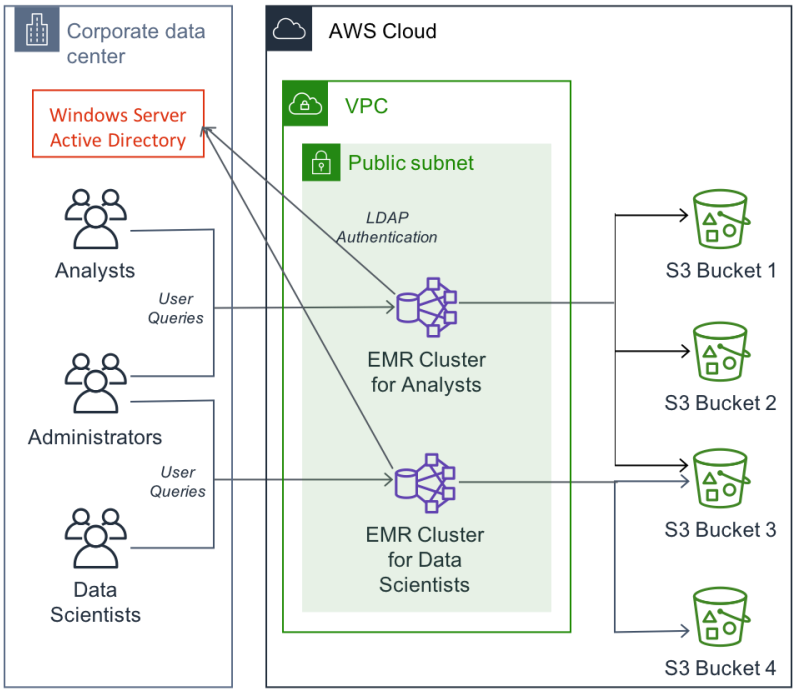


Figure 15: Separate EMR clusters

Use Case

- There is small number of end user groups, all of whom share the same level of permissions to the same data.
- The data being accessed is not highly sensitive data.

Implementation

1. Create different EC2 Instance Roles for each EMR cluster that represents the permissions you want end users to have access to.
2. Enable LDAP authentication to the applications that the end users are using, like Hue and Zeppelin. You want Hue and Zeppelin to not impersonate the user. If impersonation is required, you need to either manually create users and groups, or set up SSSD on all nodes and go to your on premises LDAP.
3. Disable SSH to the cluster.

Data Migration

Using Amazon S3 as the Central Data Repository

Many customers on AWS use Amazon Simple Storage Service (Amazon S3) as their central repository, more commonly referred to as a data lake, to securely collect, store, and analyze their data at a massive scale.[2] Organizations with data warehouses, are realizing the benefits of data lakes to enable diverse query capabilities, data science use cases, and advanced capabilities for discovering new information models. To build their data lakes on AWS, customers often use Amazon S3 as the central storage repository instead of a Hadoop Distributed File System (HDFS).

Amazon S3 is built to store and retrieve any amount of data, with unmatched availability, and can deliver 99.999999999% (11 9's) of durability. Amazon S3 provides comprehensive security and compliance capabilities that meet even the most stringent regulatory requirements.

The following figure shows a high-level overview of the data lake architectural pattern, with Amazon S3 as the central storage repository.

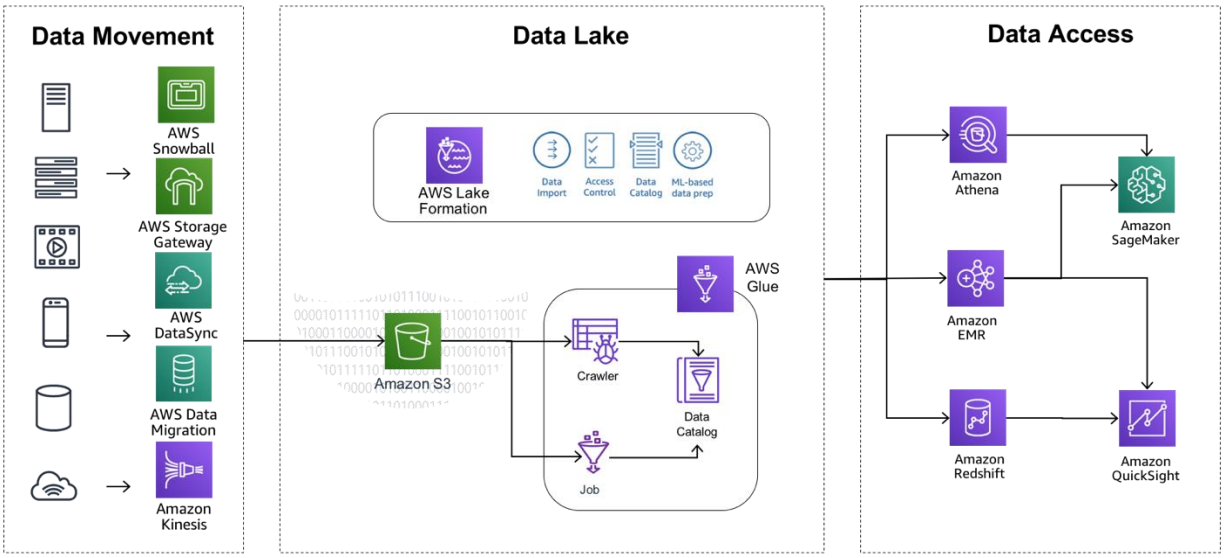


Figure 16: Data lake architecture with Amazon S3

Benefits of using Amazon S3

Some of the key benefits of using Amazon S3 as the storage platform include:

- **Decoupling of storage from compute and data processing** – In traditional Apache Hadoop and data warehouse solutions, storage and compute resources are tightly coupled, making it difficult to optimize costs and data processing workflows. With your data in Amazon S3, you can launch as much or as little compute capacity as you need using Amazon Elastic Compute Cloud (Amazon EC2), and you can use AWS analytics services to process your data. You can optimize your EC2 instances to provide the right ratios of CPU, memory, and bandwidth for best performance and even leverage Amazon EC2 Spot Instances.
- **Centralized data architecture** – Amazon S3 makes it easy to build a multi-tenant environment, where many users can bring their own data analytics tools to a common set of data. This approach improves both cost and data governance over that of traditional solutions, which require multiple copies of data to be distributed across multiple processing platforms.
- **Integration with AWS services** – Use Amazon S3 with Amazon Athena, Amazon Redshift Spectrum, Amazon Rekognition, and AWS Glue to query and process data. Amazon S3 also integrates with AWS Lambda serverless computing to run code without provisioning or managing servers. With all of these capabilities, you only pay for the data you process and for the compute time that you consume.
- **Standardized APIs** – Amazon S3 RESTful APIs are simple, easy to use, and supported by most major third-party independent software vendors (ISVs), including leading Apache Hadoop and analytics tool vendors. This enables customers to bring the tools they are most comfortable with and knowledgeable about to help them perform analytics on data in Amazon S3.

Migrating Your Data from On-premises

The first step to building data lakes on AWS is to move data to the cloud. When migrating the data to Amazon S3, the physical limitations of bandwidth and transfer speeds may restrict the ability to move data without major disruption, high costs, and time.

To help customers move their data from their data center to AWS, AWS provides the following options:

- The ability to move data between your on-premises storage and AWS storage services or between AWS storage services using AWS DataSync when bandwidth is available.
- The ability to move petabytes to exabytes of data to AWS using AWS Snowball and AWS Snowmobile appliances.
- The ability to move large quantities of data over a dedicated network connection between your data center and AWS with AWS Direct Connect.
- The ability to move data in real time from relational databases using AWS Data Migration Service, and from internet sources such as websites and mobile apps using Amazon Kinesis.

Petabytes to Exabytes of Data on a One-time Basis

AWS DataSync

AWS DataSync is an online data transfer service that simplifies, automates, and accelerates the process of moving data between your on-premises storage and AWS storage services or between AWS storage services. DataSync supports Network File System (NFS) shares, Server Message Block (SMB) shares, Hadoop Distributed File Systems (HDFS), self-managed object storage, AWS Snowcone, Amazon Simple Storage Service (Amazon S3), Amazon Elastic File System (Amazon EFS), Amazon FSx for Windows File Server, Amazon FSx for Lustre, Amazon FSx for OpenZFS, and Amazon FSx for NetApp ONTAP as well as Google Cloud Storage and Azure Files storage locations.

DataSync can help you accomplish the following tasks:

- AWS DataSync can copy data between Hadoop Distributed File Systems (HDFS) and AWS Storage
- AWS DataSync can now copy data to and from Amazon FSx for Lustre
- AWS DataSync can now copy data to and from Amazon FSx for OpenZFS
- AWS DataSync now supports Google Cloud Storage and Azure Files storage locations
- AWS DataSync adds support for Amazon EFS security features
- AWS DataSync can now copy data to and from Amazon FSx for NetApp ONTAP
- AWS DataSync can copy data to and from Amazon S3
- AWS DataSync can copy data to and from Amazon FSx for Windows

AWS Snow Family

AWS Snowball is a petabyte-scale data transport solution that uses secure devices to transfer large amounts of data into and out of AWS. You can transfer data to AWS Snowball, through the Snowball Client, which is a standalone terminal application that runs on a local workstation to do data transfer.

AWS Snowball Edge is a type of Snowball device with on-board storage and compute power for select AWS capabilities. Snowball Edge can do local processing and edge-computing workloads in addition to transferring data between your local environment and the AWS Cloud. For more information on differences between AWS Snowball and AWS Snowball Edge, see AWS Snowball Use Case Differences. For more information on best practices in migrating data using AWS Snow Family, see the AWS Big Data Blog post: Migrate HDFS files to an Amazon S3 data lake with AWS Snowball Edge & Data Migration Best Practices with AWS Snowball Edge.

Customers who are using Hadoop Distributed File System (HDFS) can also copy data directly from HDFS to an AWS Snowball using the Snowball Client. To copy data from HDFS to Snowball, you must install Snowball Client on an on-premises host that can access the HDFS cluster, and then copy files from HDFS to S3 via Snowball. Additionally, the S3 SDK Adapter for Snowball provides an S3 compatible interface to the Snowball Client for reading and writing data on a Snowball. Customers who prefer a tighter

integration can use the S3 Adapter to easily extend their existing applications and workflows to seamlessly integrate with Snowball.

We recommend that you use AWS Snowmobile to migrate large datasets of 10 PB or more in a single location. For datasets less than 10 PB or distributed in multiple locations, use AWS Snowball. Make sure to evaluate the amount of available bandwidth in your network backbone. If you have a high-speed backbone with hundreds of giga-bytes/second of spare throughput, then you can use Snowmobile to migrate the large datasets all at once. If you have limited bandwidth on your backbone, you should consider using multiple Snowballs to migrate the data incrementally.

> **Note:** As a general rule, if it takes more than one week to upload your data to AWS using the spare capacity of your existing internet connection, then you should consider using Snowball. For example, if you have a 100-Mb connection that you can solely dedicate to transferring your data and need to transfer 100 TB of data, it will take more than 100 days to complete data transfer over that connection. On the other hand, the same amount of data can be transferred in about a week, by using multiple Snowballs.

For more information on when to use Snowball for data transfer, see AWS Snowball FAQs.

Large Quantities of Data on an Ongoing Basis

AWS Direct Connect and VPN Connection

AWS Direct Connect provides you with dedicated, fast connections from your premises to the AWS network. AWS Direct Connect may be the correct choice if you need to transfer large quantities of data to AWS on an ongoing basis. AWS Direct Connect lets you establish 1 Gbps or 10-Gbps dedicated network connections (or multiple connections) between AWS networks and one of the AWS Direct Connect locations. It uses industry-standard VLANs, which enable you to access resources running within an Amazon Virtual Private Cloud (Amazon VPC) using private IP addresses.

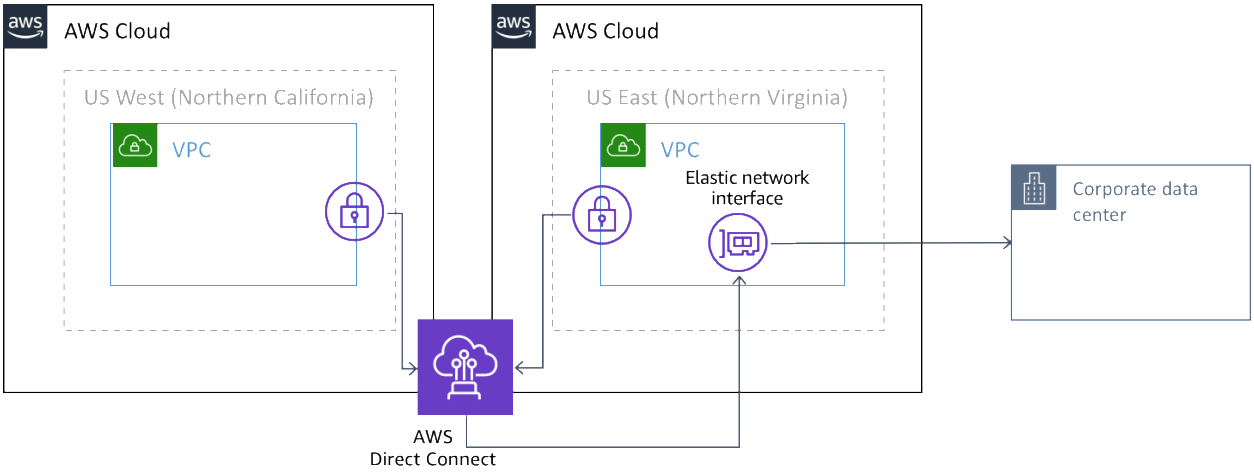


Figure 17: AWS Direct Connect

In addition to AWS Direct Connect, you can also enable communication between your remote network and your VPC by creating an encrypted connection known as a VPN connection. A VPN connection is created using Internet Protocol security (IPsec) over the internet. You can create a VPN connection by attaching a virtual private gateway to the VPC, creating a custom route table, updating your security group rules, and creating an AWS managed VPN connection.

> **Note:** We recommend that you use AWS Direct Connect for large ongoing data transfer needs, since AWS Direct Connect can reduce costs, increase bandwidth, and provide a more consistent network experience than internet-based VPN connections. VPN connections are a good solution if you have an immediate need, have low to modest bandwidth requirements, and can tolerate the inherent variability in internet-based connectivity.

Within the connection established between your on-premises environment using either of these methods, you can use AWS Direct Connect to easily migrate your data into Amazon S3 on an ongoing basis, using any of the following approaches.

Accessing Amazon S3 from Hadoop

Apache Hadoop has a connector for Amazon S3 (referred to as S3N or S3A), which can be leveraged with the DistCp tool to migrate the data from HDFS. The command to transfer data typically looks like the following:

```
hadoop distcp hdfs://source-folder s3a://destination-bucket
```

Often, the reason for the migration is a lack of compute capacity in the on-premises cluster. Customers in that situation leverage the S3DistCp tool provided by Amazon EMR to pull the data from HDFS onto Amazon S3. For more information on best practices in this scenario, see the AWS Big Data Blog post Seven Tips for Using S3DistCp on Amazon EMR to Move Data Efficiently Between HDFS and Amazon S3. You can leverage a commercially available solution such as WANDisco Fusion and Cloudera BDR to move data from HDFS onto Amazon S3.

You can also leverage Apache Hadoop and Amazon EMR integration with Amazon S3, and have the data processing workflows write directly to Amazon S3. For example, you can run Apache Sqoop jobs on an Amazon EMR cluster to extract data from a relation database and write it to Amazon S3.

AWS Glue

AWS Glue is a fully managed extract, transform, and load (ETL) service that makes it easier to prepare and load your data for analytics. AWS Glue can also discover your data and stores the associated metadata (for example, a table definition and schema) in the AWS Glue Data Catalog. AWS Glue is designed to simplify the tasks of moving and transforming your datasets for analysis. It's a serverless, fully managed service built on top of the popular Apache Spark execution framework.

AWS Glue can access on-premises relational databases via Java Database Connectivity (JDBC) to crawl a data store and catalog its metadata in the AWS Glue Data Catalog. The connection can be also used by any ETL job that uses the data store as a source or target, like writing the data back to Amazon S3. The figure below illustrates the workflow to extract data from a relational database, transform the data, and store the results in Amazon S3.

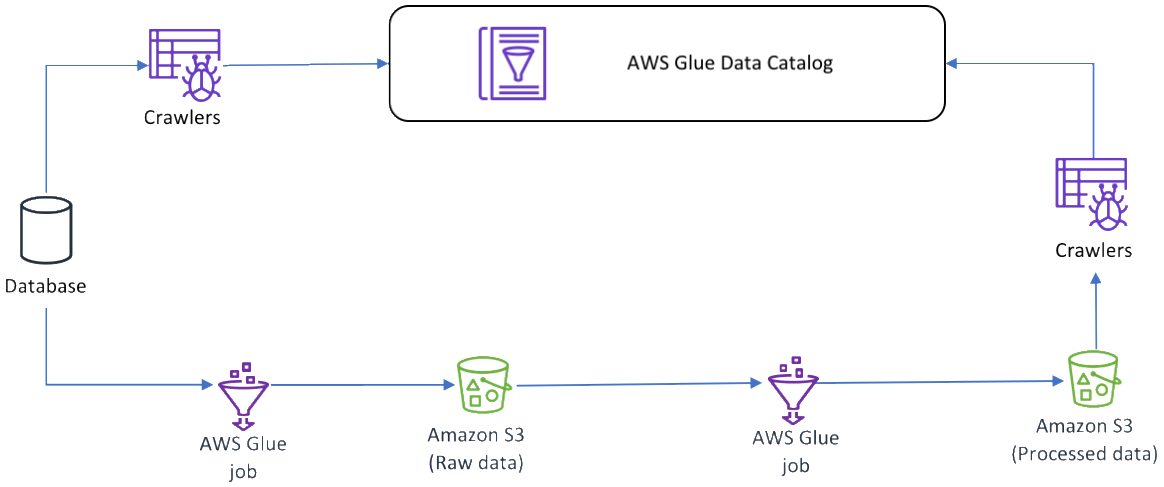


Figure 18: AWS Glue Data Catalog

For more information, see the AWS Big Data Blog post How to extract, transform, and load data for analytic processing using AWS Glue.

AWS DataSync

AWS DataSync supports Hadoop Distributed File Systems (HDFS). The most common way to get data onto a cluster is to upload the data to Amazon S3 and use the built-in features of Amazon EMR to load the data onto your cluster. To upload data from an on-premises Hadoop cluster to your S3 bucket, you first deploy one or more DataSync agents in the same network as your on-premises storage. An agent is a virtual machine (VM) that is used to read data from or write data to a self-managed location. You then activate your agents in the AWS account and AWS Region where your S3 bucket is located.

After your agent is activated, you create a source location for your HDFS location, a destination location for your S3 bucket, and a task. A task is a set of two locations (source and destination) and a set of default options that you use to control the behavior of the task. Finally, you run your DataSync task to transfer data from the source to the destination.

> **Note:** We recommend that you use AWS DataSync to rapidly copy data out from your Hadoop Cluster into Amazon S3. You can use DataSync for fast transfer of existing data to Amazon S3, and the File Gateway configuration of Storage Gateway for subsequent low-latency access to this data from on-premises applications. Learn more about how to use AWS DataSync to move from Hadoop to Amazon S3 in this blog.

AWS Storage Gateway

AWS Storage Gateway enables you to integrate legacy on-premises applications with Amazon S3, and is used for backup, tiering, and local access to objects stored in Amazon S3. The File Gateway configuration of Storage Gateway offers on-premises devices and applications low-latency access to the data in Amazon S3 via an NFS connection. This means that you can easily integrate applications and platforms that don't have native Amazon S3 capabilities — such as on-premises lab equipment, mainframe computers, databases, and data warehouses — to directly write their files into Amazon S3. Files written to this mount point are converted to objects stored in Amazon S3 in their original format without any proprietary modification.

Event and Streaming Data on a Continuous Basis

AWS Database Migration Service

AWS Database Migration Service (AWS DMS) helps you migrate databases to AWS easily and securely. AWS DMS performs continuous data replication using change data capture (CDC). Most database management systems manage a transaction log that records changes made to the database contents and to metadata. By using engine-specific API operations and functions, AWS DMS reads the transaction log and captures the changes made to the database in a non-intrusive manner.

You can use AWS DMS to migrate data from any of the supported database sources to Amazon S3. When using Amazon S3 as a target in an AWS DMS task, both full load and change data capture (CDC) data is written to comma-separated-values (CSV) format. For more information on how to query this data, see the AWS Database Blog post on Using AWS Database Migration Service and Amazon Athena to Replicate and Run Ad Hoc Queries on a SQL Server Database.

> **Note:** For use cases that require a database migration from on-premises to AWS or database replication between on-premises sources and sources on AWS, we recommend that you use AWS DMS. Once your data is in AWS, you can use AWS Glue to move and transform the data.

Amazon Kinesis Data Firehose

Amazon Kinesis Data Firehose is a fully managed service for delivering real-time streaming data directly to Amazon S3. Amazon Kinesis Data Firehose automatically scales to match the volume and throughput of streaming data, and requires no ongoing administration. Amazon Kinesis Data Firehose can concatenate multiple incoming records, and then deliver them to Amazon S3 as a single S3 object. This is an important capability because it reduces Amazon S3 transaction costs and transactions per second load.

Amazon Kinesis Data Firehose can also be configured to transform streaming data before it's stored in Amazon S3. Its transformation capabilities include compression, encryption, data batching, and Lambda functions. Amazon Kinesis Data Firehose can compress data before it's stored in Amazon S3. It currently supports GZIP, ZIP, and SNAPPY compression formats. See the following diagram for an example workflow.

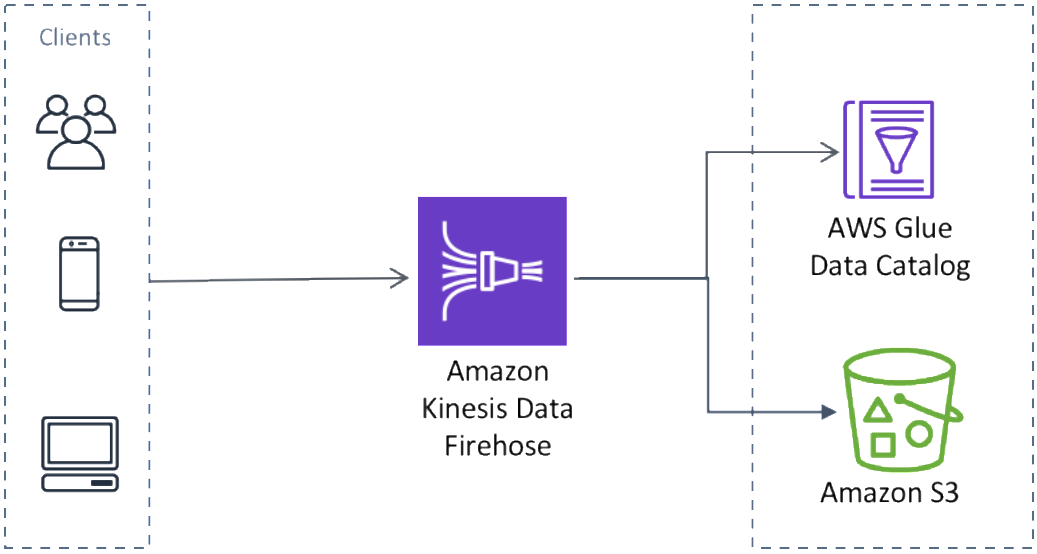


Figure 19: Amazon Kinesis Data Firehose Architecture

In *Figure 19*, an agent writes data from the source into an Amazon Kinesis Data Firehose. An example is Amazon Kinesis Agent, which can write data from a set of files to an Amazon Kinesis Data stream. Once the event is streamed into an Amazon Kinesis Data Firehose Data Stream, the Kinesis Data Firehose delivers the data to a configured Amazon S3 bucket. You can also configure a Kinesis Data Firehose to deliver data into Apache Parquet format using the schema from AWS Glue Data Catalog. For another example, see the AWS Database Blog post on Streaming Changes in a Database with Amazon Kinesis.

Optimizing an Amazon S3-Based Central Data Repository

By using Amazon S3 as the central repository for your data lake, you have already optimized the total cost of ownership (TCO) by decoupling storage and compute, and paying for compute based on the

actual need and usage. AWS and Amazon S3 have several features that can optimize the storage footprint in your data lake to further reduce costs. By using columnar file formats, you can speed up your queries while reducing costs. In this section, we look at a couple different optimizations you can do with your Amazon S3-based central data repository.

Optimizing Storage Cost

Amazon S3 offers a range of storage classes designed for both frequently and infrequently accessed data. In addition, customers can use Amazon S3 Glacier for long-term archive. Amazon S3 offers configurable lifecycle policies for managing your data throughout its lifecycle. The next section covers options available in AWS for optimizing your storage cost by tiering your data across Amazon S3 storage classes and Amazon S3 Glacier.

Amazon S3 Storage Classes

Amazon S3 Standard (S3 Standard) offers high durability, availability, and performance object storage for frequently accessed data. Because it delivers low latency and high throughput, S3 Standard is the right choice for storing your data that is accessed frequently, like big data analytics.

Amazon S3 Intelligent-Tiering (S3 Intelligent-Tiering) offers the same low latency and high throughput performance of S3 Standard, but enables you to further optimize storage costs by automatically moving data between two tiers - one tier that is optimized for frequent access and another lower-cost tier that is optimized for infrequent access. For a small monthly monitoring and automation fee per object, Amazon S3 monitors access patterns of the objects in S3 Intelligent-Tiering, and moves the ones that have not been accessed for 30 consecutive days to the infrequent access tier. It is the ideal storage class for long-lived data with access patterns that are unknown or unpredictable.

Amazon S3 Standard-Infrequent Access (S3 Standard-IA) offers the high durability, high throughput, and low latency of S3 Standard, with a low per GB storage price and per GB retrieval fee. S3 Standard-IA is ideal for data that is accessed less frequently, but requires rapid access when needed.

Amazon S3 One Zone-Infrequent Access (S3 One Zone-IA) stores data in a single Availability Zone (AZ) and offers a lower-cost option (20% less than storing it in S3 Standard-IA) for infrequently accessed data that does not require the availability and resilience of S3 Standard or S3 Standard-IA storage classes. S3 One Zone-IA is ideal for storing backup copies or easily re-creatable data.

Amazon S3 Glacier and Amazon S3 Glacier Deep Archive

Amazon S3 Glacier and Amazon S3 Glacier Deep Archive are a low-cost storage service that provides durable storage with security features for data archiving and backup. They have the same data durability (99.999999999%) as Amazon S3 and the same integration with AWS security features. They can be integrated with Amazon S3 by using Amazon S3 lifecycle management on data assets stored in Amazon S3, so that data assets can be seamlessly migrated from Amazon S3 to Amazon S3 Glacier or Glacier Deep Archive. Amazon S3 Glacier is a great long-term storage choice when the portions of the data

might need to be retrieved in minutes. On the other hand, Amazon S3 Glacier Deep Archive is another great long-term storage choice when archived data rarely needs to be accessed.

Amazon S3 Lifecycle Management

Amazon S3 lifecycle management enables you to create data lifecycle rules, which can be used to automatically migrate data assets to a lower-cost tier of storage, such as Amazon S3 Standard - Infrequent Access or Amazon S3 Glacier, or let them expire when they are no longer needed. A lifecycle configuration, which consists of an XML file, comprises a set of rules with predefined actions that you want Amazon S3 to perform on data assets during their lifetime. Lifecycle configurations can perform actions based on data asset age and data asset names, but can also be combined with S3 object tagging to perform granular management of data assets.

Amazon S3 Storage Class Analysis

One of the challenges of developing and configuring lifecycle rules for the data lake is gaining an understanding of how data assets are accessed over time. It makes economic sense to transition data assets to a more cost-effective storage or archive tier if those objects are infrequently accessed. Otherwise, data access charges associated with these more cost-effective storage classes could negate any potential savings. Amazon S3 provides Amazon S3 storage class analysis to help you understand how data lake data assets are used. Amazon S3 storage class analysis uses machine learning algorithms on collected access data to help you develop lifecycle rules that optimize costs.

Tiering Your Data Lake Storage

A data lake generally has raw data being ingested from many sources, which is then transformed and optimized for ad hoc querying and on-going analysis. Many advanced uses, such as machine learning and artificial intelligence, consist of building data models and then training and refining these models using the raw historical data. In addition, by keeping the historical raw data, you can go back and reprocess historical data to provide new insights in the transformed data. Since these historical data assets are infrequently accessed and may be large in size, they are often well suited to be stored in the S3 Standard-IA storage class, which is optimized for infrequent data access. This data can then be moved to Amazon S3 Glacier for long-term archival using lifecycle policies.

Another tiering approach you can use in a data lake is to keep processed data and results, as needed for compliance and audit purposes, in an Amazon S3 Glacier vault. Amazon S3 Glacier Vault Lock allows data lake administrators to easily deploy and enforce compliance controls on individual Glacier vaults via a lockable policy. Administrators can specify controls such as “write once read many” (WORM) in a vault lock policy and lock the policy from future edits. Once locked, the policy becomes immutable and Amazon S3 Glacier enforces the prescribed controls to help achieve your compliance objectives, and provide an audit trail for these assets using AWS CloudTrail.

Optimizing Cost and Performance

You can also use optimize your use of Amazon S3 around cost and performance. Amazon S3 provides a high-performance foundation for the data lake because its enormous scale provides virtually limitless throughput and high transaction rates. Using Amazon S3 best practices for data asset naming ensures high levels of performance. For more information on best practices see What is Amazon S3 in the *Amazon Simple Storage Service Developers Guide*.

Partition the Data

Partitioning is an important technique for organizing datasets so they can be queried efficiently. Partitioning organizes data in a hierarchical directory structure based on the distinct values of one or more columns. For example, you may decide to partition your application logs in Amazon S3 by date, broken down by year, month, and day. Log files that correspond to a single day's worth of data are then placed under a prefix such as `s3://my_bucket/logs/year=2018/month=01/day=23/`. Services such as Amazon Athena, Amazon Redshift Spectrum, and AWS Glue can use this layout to filter data by partition value without having to read all of the underlying data from Amazon S3.

Use Columnar Data Formats

Another area of optimization is to use an optimal data format for storing the transformed and normalized dataset. Columnar formats can compress data better thereby reducing the storage cost, and also substantially increase query performance. Apache Parquet and Apache ORC are two such columnar formats, which can reduce the required storage footprint, improve query performance, and reduce querying costs for AWS services such as Amazon Athena and Amazon Redshift Spectrum, which charge by amount of data scanned.

Bias Toward Large File Sizes

Big data processing runs more efficiently when the tasks can be parallelized over multiple files, each of which can be further split to increase the parallelism further. However, if your files are too small (generally fewer than 64 MBs), the processing engine may be spending additional time with the overhead of opening Amazon S3 files, listing prefixes, getting object metadata, setting up data transfer, reading file headers, and reading compression dictionaries. If your file cannot be split and the files are too large, the query processing waits until a single reader has completed reading the entire file. This can reduce parallelism.

One remedy to solve your small file problem is to use the S3DistCP utility on Amazon EMR. You can use it to combine smaller files into larger objects. You can also use S3DistCP to move large amounts of data in an optimized fashion from HDFS to Amazon S3, Amazon S3 to Amazon S3, and Amazon S3 to HDFS.

Using AWS Glue to Transform and Normalize Data

AWS Glue jobs help you transform and normalize data to optimize query performance. AWS Glue supports writing to both Apache Parquet and Apache ORC formats, which can make it easier and faster for you to transform data to an optimal format for Amazon Athena. Additionally, AWS Glue crawlers infer file types and schemas, and automatically identify the partition structure of your dataset when they populate the AWS Glue Data Catalog. The resulting partition columns are available for querying in AWS Glue ETL jobs or query engines like Amazon Athena.

The following diagram shows how to transform ingested data in CSV format into Apache Parquet for querying by Amazon Athena with improved performance and reduced cost. For more information on this architecture, see Build a Data Lake Foundation with AWS Glue and Amazon S3.

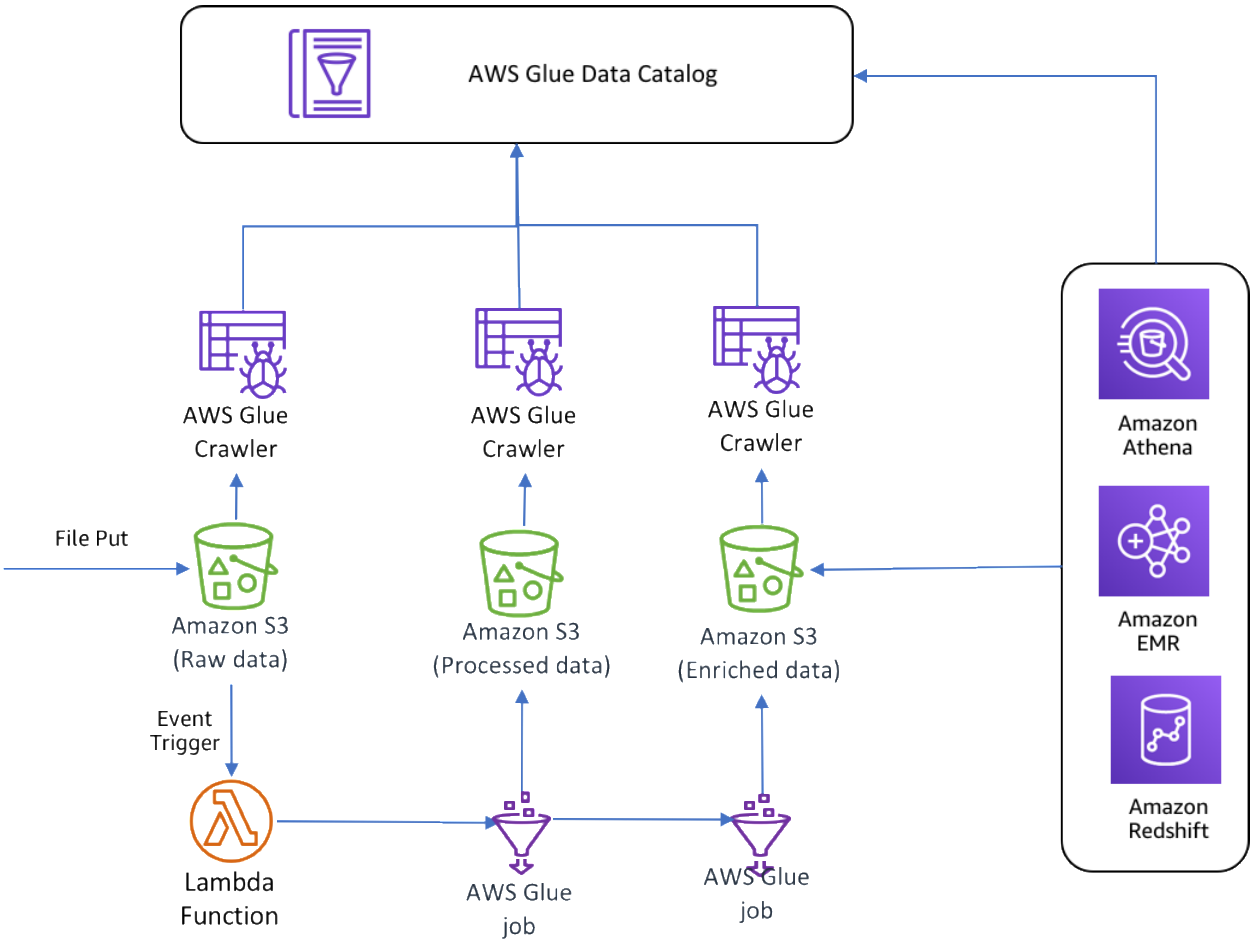


Figure 20: Transforming data with AWS Glue

Understand How Applications Work with Amazon S3

As we highlighted earlier in this whitepaper, applications such as Apache Hive, Apache Spark, and MapReduce work with Amazon S3 by mapping the HDFS APIs to Amazon S3 APIs (like EMRFS available

with Amazon EMR). Although for most of the APIs, there is a direct mapping between the HDFS APIs and Amazon S3, some of the file system APIs, such as rename and seek, do not have an equivalent API in Amazon S3, and are mapped in an indirect manner. Further, semantic differences exist between Amazon S3, an object store, HDFS, and a file system, such as prefixes versus directories. The overview section of the Hadoop integration with AWS documentation provides a good introduction to such semantic differences.

Because of these indirect mappings or semantic differences, when using the open source applications on Amazon EMR, you may sometimes see performance issues or job failures. Many applications also require specific customizations to work with Amazon S3, such as those in Hive Blobstore Optimizations or Amazon S3 Optimized Committer, available with Spark on Amazon EMR. The application-specific sections in this guide also provide you with some of these considerations and how to optimize them to work with Amazon S3.

Data Catalog Migration

Apache Hive is an open source data warehouse and analytics package that runs on top of an Apache Hadoop cluster. Hive is one of the applications that can run on Amazon EMR. A Hive metastore contains a description of the table and the underlying data on which it is built, including the partition names and data types. There are three modes for Hive metastore deployment: embedded metastore, local metastore, and remote metastore. When migrating a Hadoop on-premises cluster to Amazon EMR, you must follow a different strategy depending on how the Hive metastore is being deployed. This chapter provides the different patterns used to deploy a Hive metastore and how to migrate an existing metastore to Amazon EMR.

Hive Metastore Deployment Patterns

Apache Hive ships with the Derby database, an embedded database backed by local disk. This database is used for embedded metastores and but we do not recommend that you use Derby as it cannot scale for production-level workloads. In Amazon EMR, by default, Hive records metastore information in a MySQL database on the master node's file system. When a cluster terminates, all cluster nodes are shut down, including the master node. When this happens, local data is lost because the node's file systems use ephemeral storage. To avoid this scenario, we recommend that you create an external Hive metastore outside of the cluster. There are two options for an Amazon EMR external metastore:

- AWS Glue Data Catalog
- Amazon RDS database or Amazon Aurora database

For an FAQ on these options, refer to Appendix D: Data Catalog Migration FAQs.

AWS Glue Data Catalog

The AWS Glue Data Catalog provides a unified metadata repository across a variety of data sources and data formats, integrating with Amazon EMR, Amazon RDS, Amazon Redshift, Redshift Spectrum, Amazon Athena, and any application compatible with the Apache Hive metastore. The benefits of using the AWS Glue Data Catalog are that you don't have to manage the Hive metastore database instance separately, don't have to maintain ongoing replication, and don't have to scale up the instance. An AWS Glue Data Catalog is fully managed and serverless, highly available, fault-tolerant, maintains data replicas to avoid failure, and expands hardware depending on the usage. When creating a new Amazon EMR cluster, you can choose an AWS Glue Data Catalog as the Hive metastore. (**Note:** This option is only available on Amazon EMR version 5.8.0 or later.)

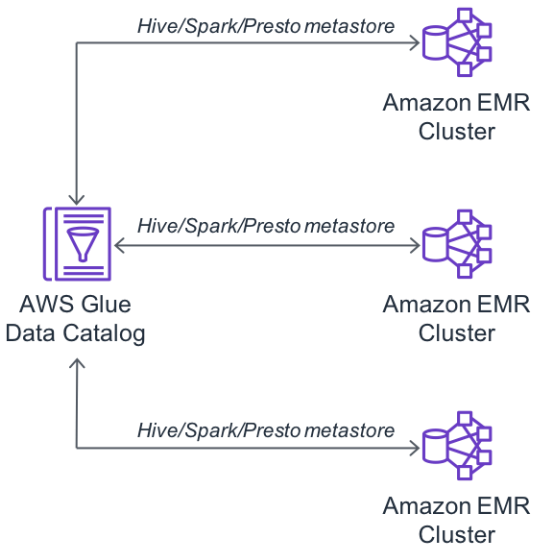


Figure 21: Using AWS Glue Data Catalog as the Hive metastore

Configuration within Amazon EMR Console

Apache Hive, Presto, and Apache Spark all use Hive metastore. Within the Amazon EMR console, you have the option to use AWS Glue Data Catalog for any of those three applications. The following figure shows this setting as it appears in the console under Advanced Options.

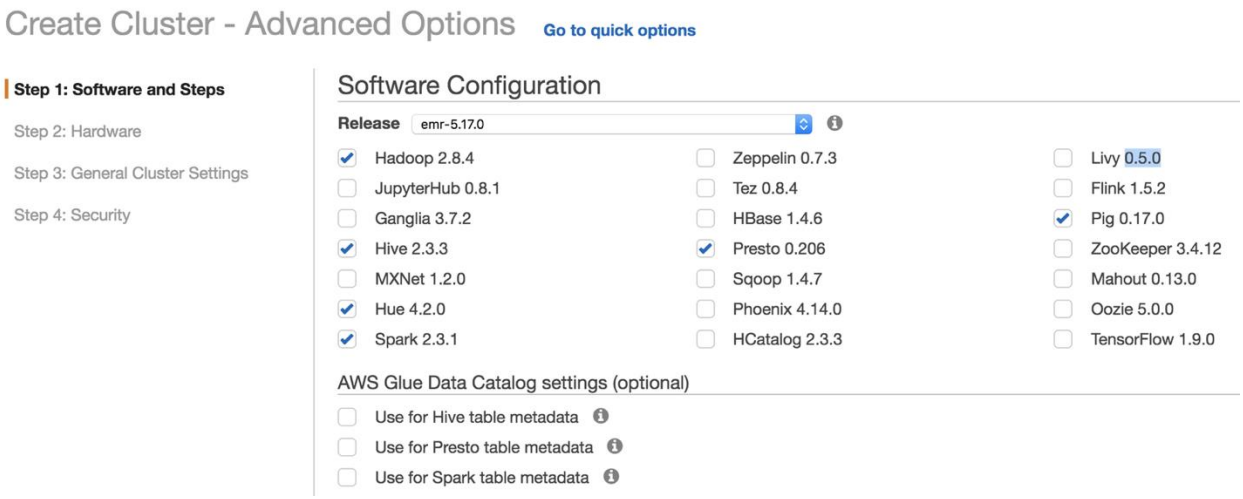


Figure 22: AWS Glue Data Catalog settings in Amazon EMR

Configuration within AWS CLI or Amazon EMR API

The Hive site configuration file must reflect an AWS Glue Data Catalog for Hive metastore. To do that, point the `hive.metastore.client.factory.class` property to `AWSGlueDataCatalogHiveClientFactory`, like below:

```
[
  {
    "Classification": "hive-site",
    "Properties": {
      "hive.metastore.client.factory.class":
"com.amazonaws.glue.catalog.metastore.AWSGlueDataCatalogHiveClientFactory"
    }
  }
]
```

This setting can be passed as part of application configuration when creating an Amazon EMR cluster. See Configuring Applications for more details about how to pass application configurations.

Considerations

- You can enable encryption for an AWS Glue Data Catalog. For details, see Setting Up Encryption in AWS Glue.
- Column statistics, Hive authorizations, and Hive constraints are not currently supported. To see a list of AWS Glue Data Catalog's constraints, see Using the AWS Glue Data Catalog as the Metastore for Hive.
- An AWS Glue Data Catalog has versions, which means a table can have multiple schema versions. AWS Glue stores that information in AWS Glue Data Catalog, including the Hive metastore data.

Amazon RDS or Amazon Aurora

There are two main steps to deploy an external Hive metastore:

1. Create an Amazon RDS (MySQL database) or Amazon Aurora database.
2. Configure the `hive-site.xml` file to point to MySQL or Aurora database.

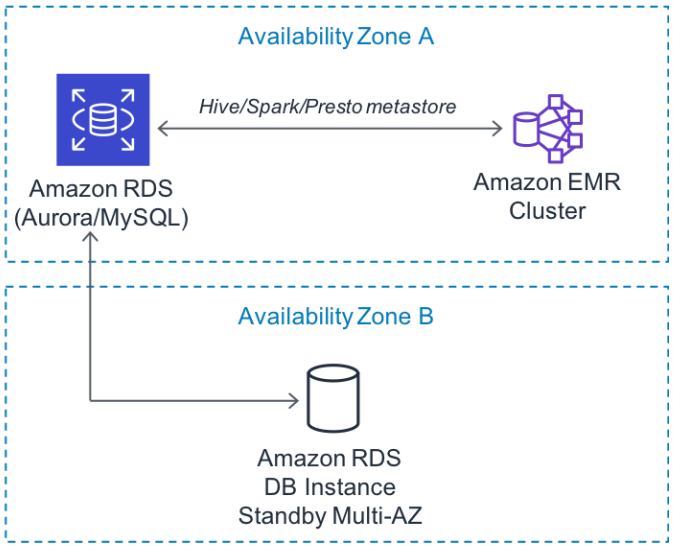


Figure 23: Creating Hive metastore on Amazon RDS or Amazon Aurora

Create a MySQL or Amazon Aurora Database

1. Create the database.
    - To create a MySQL database on Amazon RDS, see Create and Connect to a MySQL Database with Amazon RDS.
    - To create an Amazon Aurora database, see Creating an Amazon Aurora DB Cluster.
2. Note the URL, user name, password, and database name.
3. Update the security group to allow connections between the Amazon EMR cluster and a MySQL database.

Configure Amazon EMR for External Hive Metastore

1. Create a configuration file containing hive-site classification info as given below.
    - `javax.jdo.option.ConnectionDriverName` should reflect the driver `org.mariadb.jdbc.Driver` (preferred driver)
    - Point the following three settings to the newly created database:
      `javax.jdo.option.ConnectionURL`
      `javax.jdo.option.ConnectionUserName`
      `javax.jdo.option.ConnectionPassword`

```
[
    {
      "Classification": "hive-site",
      "Properties": {
        "javax.jdo.option.ConnectionURL":
"jdbc:mysql:\/\/hostname:3306\/hive?createDatabaseIfNotExist=true",
        "javax.jdo.option.ConnectionDriverName":
"org.mariadb.jdbc.Driver",
        "javax.jdo.option.ConnectionUserName": "username",
        "javax.jdo.option.ConnectionPassword": "password"
      }
    }
]
```

2. When using Amazon EMR in the console, pass this information as JSON from Amazon S3 or embedded text:

Figure 24: Configuration settings as embedded text

3. When using AWS CLI, pass the `hive-configuration.json` configuration file as a local file or from Amazon S3:

```
aws emr create-cluster --release-label emr-5.17.0 --instance-type
m4.large --instance-count 2 \
--applications Name=Hive --configurations ./hive-configuration.json
--use-default-roles
```

Figure 25: Configuration file as local file

```
aws emr create-cluster --release-label emr-5.17.0 --instance-type
m4.large --instance-count 2 \
--applications Name=Hive --configurations s3://emr-sample/hive-
configuration.json --use-default-roles
```

Figure 26: Configuration file from Amazon S3

Considerations

A Hive metastore is a single point of failure. Amazon RDS doesn't automatically replicate databases, so it's highly recommended that you enable replication when using Amazon RDS to avoid any failure. To learn more about how to create a database replica in a different Availability Zone, refer to the following sources:

- How do I create a read replica for an Amazon RDS database?
- Working with Read Replicas

Hive Metastore Migration Options

When migrating Hadoop-based workloads from on-premises to the cloud, you must migrate a Hive metastore as well. Depending on the migration plan and requirements, a metastore can be migrated in two different ways: a one-time migration that migrates an existing Hive metastore completely to AWS, or an on-going migration that migrates the Hive metastore, but keeps a copy on-premises. In this scenario, the two metastores are synced in real time during the migration phase. The following section discusses these two scenarios in detail.

One-Time Metastore Migration

This section focuses on a set of options to consider when migrating an existing Hive metastore completely to AWS. This situation is applicable to a scenario where the organization plans to use the Hive metastore on AWS. The following figure illustrates this scenario:

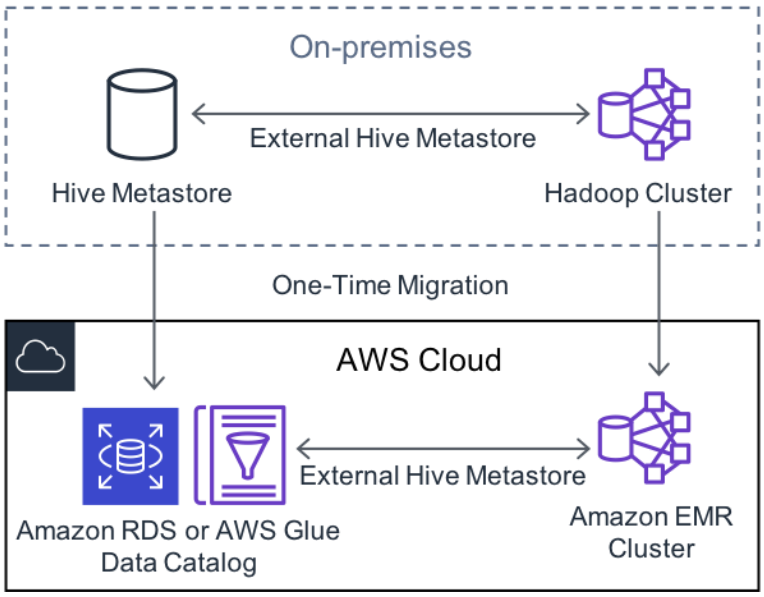


Figure 27: One-time metastore migration

Existing Hive Metastore to AWS Glue Data Catalog

In this case, the goal is to migrate existing Hive metastore from on-premises to an AWS Glue Data Catalog. You can use AWS Glue ETL job to extract metadata from your Hive metastore, and use AWS Glue jobs to load the metadata and update AWS Glue Data Catalog. See Migration between the Hive Metastore and the AWS Glue Data Catalog on GitHub document to learn more about those options:

Existing Hive Metastore to Amazon RDS

In this case, you are not leveraging an AWS Glue Data Catalog, instead, you are moving the Hive metastore data from an on-premises database to Amazon RDS. Depending on which database is currently being used to store the Hive metastore data, you need to take different steps to migrate them to the corresponding Amazon RDS instance. For example:

- MySQL on on-premises → MySQL on Amazon RDS or Amazon Aurora
- PostgreSQL on on-premises → PostgreSQL on Amazon RDS or Amazon Aurora
- Oracle on on-premises → Oracle on Amazon RDS

Here are few resources that cover how to migrate those databases to AWS:

- Migrate On-Premises MySQL Data to Amazon RDS
- Importing Data into PostgreSQL on Amazon RDS
- Migrating Data from a PostgreSQL DB Instance to an Aurora PostgreSQL DB Cluster
- Migrating Data from a MySQL DB Instance to an Amazon Aurora MySQL DB Cluster

On-going Metastore Sync

This pattern is used mainly for large-scale migrations when you want to migrate an on-premises Hive metastore to AWS, but also want to keep running the Hive metastore in your data center as well as in the cloud during the migration phase. In that case, on-going sync is required so that both Hive metastores are up-to-date. For a given time, only one application should be used for updating the Hive metastore, otherwise the metastore will be out-of-sync.

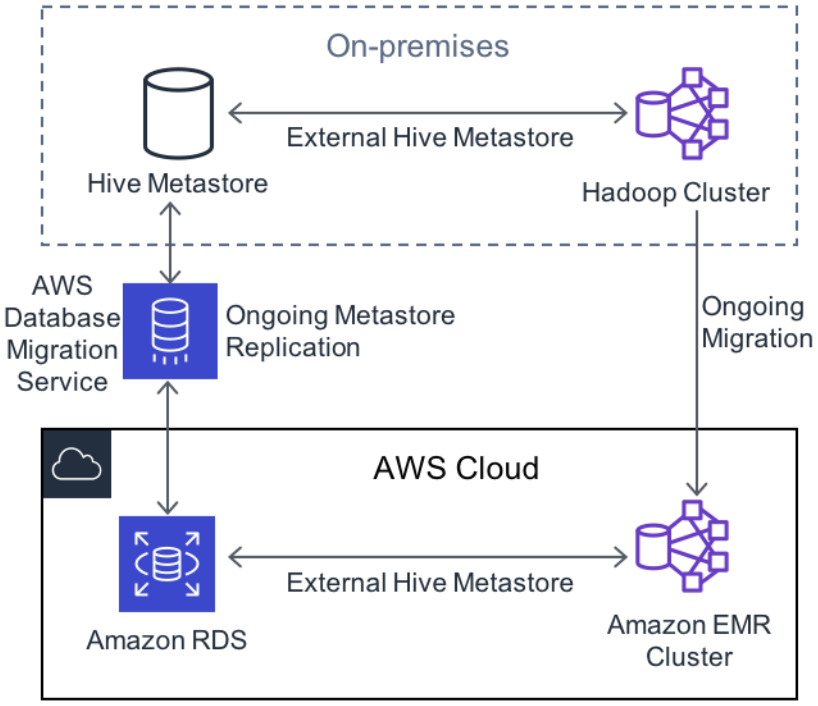


Figure 28: Ongoing metastore sync

AWS Database Migration Service is a data migration service and can be used to create on-going replication. This blog post Replicating Amazon EC2 or On-Premises SQL Server to Amazon RDS for SQL Server discusses how to achieve ongoing replication for SQL Server, but the same method applies to other databases.

Multitenancy on EMR

Amazon EMR provides a comprehensive set of features to build a highly secure multitenant Hadoop cluster resources and data. Multitenancy on Amazon EMR offers mechanisms to isolate data and resource from different tenants, and also provides controls to prevent a single application, user, and queue from monopolizing cluster resources. Multitenancy comes with its own set of challenges. For example, implementing multitenancy at all stages of a pipeline involves an understanding of the nuances of processes and tools involved; metering tenant usage of resources can be hard especially if the tenants share metadata and compute resources; scalability and time involvement can be difficult as new tenants onboard; and applying robust security controls overall can be daunting task.

This chapter discusses steps to implement multitenancy on Amazon EMR along with key dimensions, such as user, data, and resource isolation followed by recommended best practices.

There are two different conceptual models for isolating users, data, and resources when building multitenant analytics on Amazon EMR.

- Silo Mode
- Shared Mode

Silo Mode

In a silo mode, each tenant gets their own Amazon EMR cluster with specific tools for processing and analyzing their datasets. Data is stored in the tenant's Amazon S3 bucket or HDFS on the cluster. Hive metastore is typically on the cluster or stored externally on Amazon RDS.

Example Silo Scenario

The following diagram is example of a silo scenario in Amazon EMR. In this scenario, there are three different users – a data engineer, an analyst, and a data scientist – each launching their own clusters. A data engineer installs tools like Spark and Hive to manipulate and store the processed results in S3. An analyst runs tools like Spark SQL and Presto to explore datasets and send the query results to his own S3 bucket. A data scientist may also use the EMR cluster to run ML or Deep Learning frameworks.

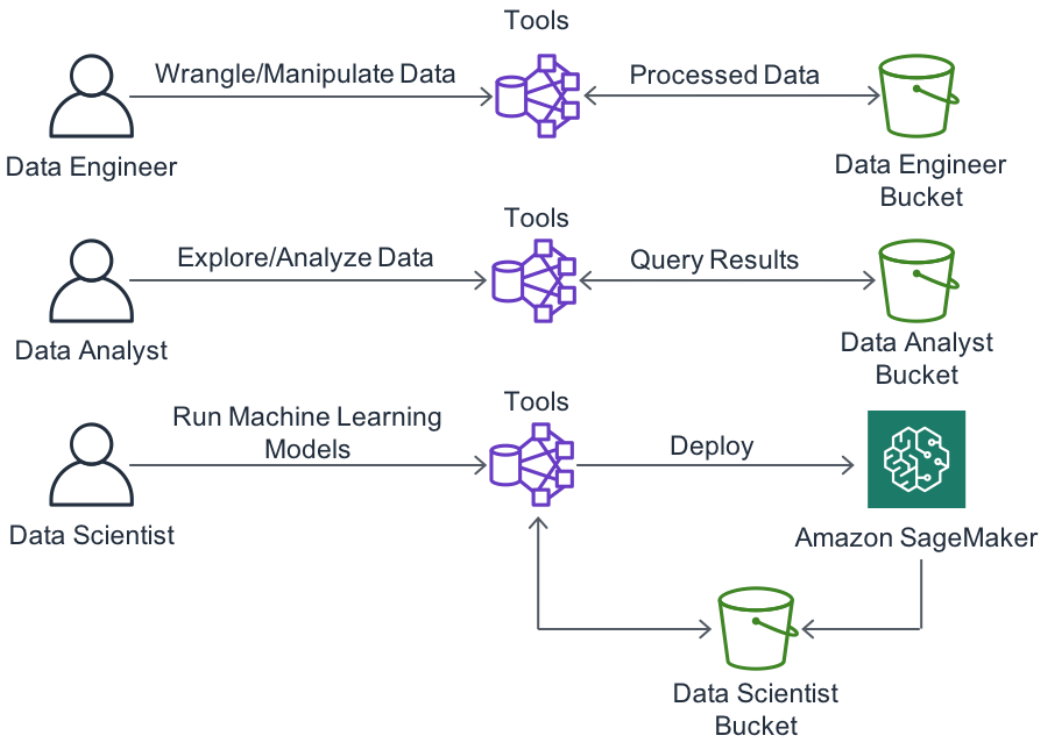


Figure 29: Example silo mode scenario

In this model, you can configure your cluster to be automatically terminated after all of the steps of your processing complete. This setup is referred to as a transient cluster. A transient cluster provides total segregation per tenant, and can also decrease costs as the cluster is charged only for the duration of the time it runs. The following table lists the advantages and disadvantages of using silo mode with Amazon EMR.

Table 3: Advantages and disadvantages of silo mode

Advantage	Disadvantage
Provides complete isolation of data and resources.	Sharing data across clusters (especially when using HDFS) can be difficult
Can be cost effective when used with Spot Instances and transient clusters.	Launching individual clusters can be expensive.
Easy to measure usage of resources per tenant.	

Shared Mode

In a shared mode, tenants share the Amazon EMR cluster with tools installed for processing/analyzing/data science – all in one cluster. Datasets are stored in the tenant's S3 bucket or

the tenant's HDFS folder on the cluster. The Hive metastore can be on the cluster or externally on Amazon RDS or AWS Glue Data Catalog. In many organizations, this shared scenario is more common. Sharing clusters between organizations is a cost-effective way of running large Hadoop installations since it enables them to derive benefits of economies of scale without creating private clusters.

A large multi-node cluster with all the tools and frameworks installed can support a variety of users. In addition, this infrastructure can also be used by end users who can launch edge nodes to run their data science platforms.

Even though it is cost effective, sharing a cluster can be a cause for concern because a tenant might monopolize resources and cause the SLA to be missed for other tenants. For example, an analyst can issue a long running query on Presto or Hive and much of the cluster resources. Or, a data scientist might train a model over large amounts of data.

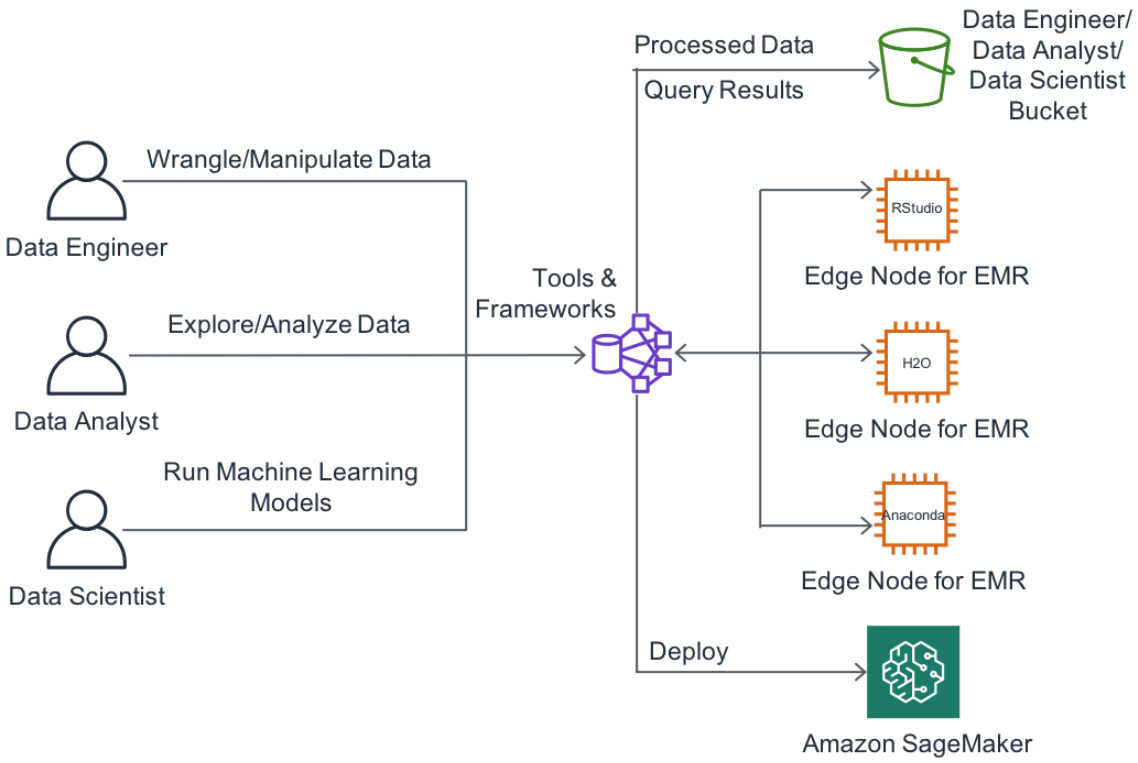


Figure 30: Example shared mode scenario

The following table lists the advantages and disadvantages of launching an Amazon EMR cluster in a shared mode.

Table 4: Advantages and disadvantages of using shared mode

Advantage	Disadvantage
Less operational burden as there is one cluster to maintain.	Hard to measure usage and resources when you have many tenants
Can be cost effective if the cluster is well-utilized.	Configuring the YARN scheduler can be difficult and complex.
	Cannot customize the cluster for individual workloads. It is not possible for data engineers and data scientists to have different configurations (instance type, instance type, volumes, etc.) for their specific workload.
	One configuration to fit all use cases. Cluster configuration is immutable and teams must work on the application optimization to improve the performance of their workload as opposed to adjusting the cluster configuration.
	Software cannot be upgraded without upgrading all applications.
	Large blast radius if something goes wrong with the cluster.

Considerations for Implementing Multitenancy on Amazon EMR

Multitenant architecture with Amazon EMR requires careful thought and planning along critical dimensions, including users, data, and resource isolation.

User Isolation

Authentication

Authentication of users is a critical piece in securing the cluster resources and preventing unauthorized access to data. On Amazon EMR, you can authenticate users through an LDAP server or set up Kerberos to provide strong authentication through secret-key cryptography or SAML Identity Provider based authentication with AWS Lake Formation integration. When you use Kerberos authentication, Amazon

EMR configures Kerberos for the applications, components, and subsystems that it installs on the cluster so that they are authenticated with each other.

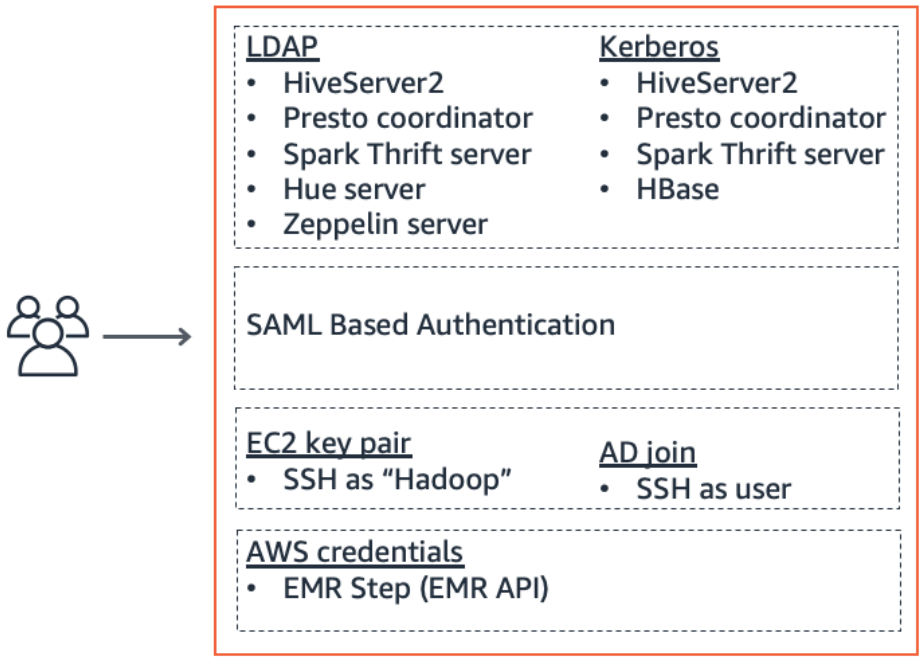


Figure 31: User authentication with LDAP, Kerberos or SAML

When you use Kerberos with Amazon EMR, you can choose to set it up as a cluster-dedicated KDC or an external KDC with different architecture options. Regardless of the architecture that you choose, you can configure Kerberos using the same following steps.

1. Create a security configuration. When you create the cluster, you must specify the security configuration and compatible cluster-specific Kerberos options. See Create Security Configurations in the *Amazon EMR Management Guide* for more details.
2. Create HDFS directories for Linux users on the cluster that match user principals in the KDC.

Once completed, you can use an Amazon EC2 key pair to authorize SSH client connections to cluster instances.

You can also use SAML Identity Provider based authentication when you integrate Amazon EMR with AWS Lake Formation. When a user is authenticated with SAML Identity Providers, Amazon EMR automatically creates a Kerberos principal and a Linux user, and jobs submitted through Spark SQL/Livy impersonate the authenticated user. For details, see Supported Applications and Features in the *Amazon EMR Management Guide*.

For example, in the following figure, user `emr-analyst` authenticated using SAML Identity Provider. Once the user is authenticated, EMR initiated a Spark Livy Session when user submitted jobs using Zeppelin or EMR Notebook.

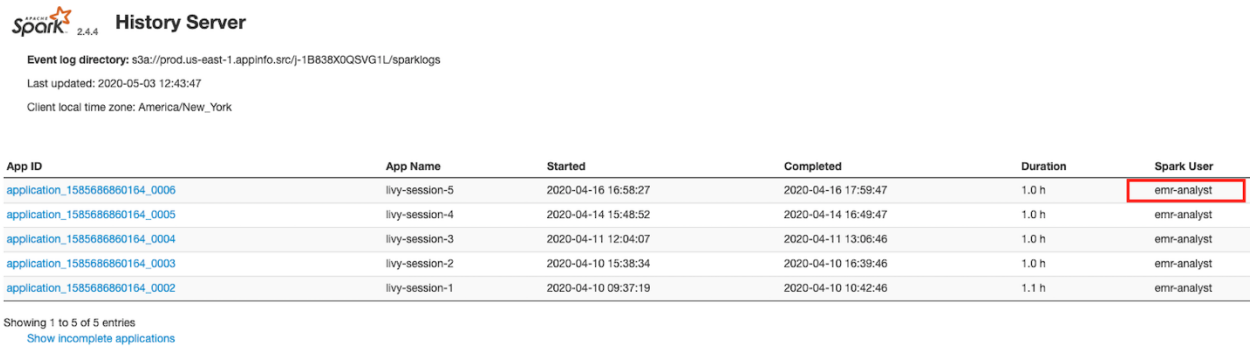


Figure 32: Spark Livy session started under authenticated user

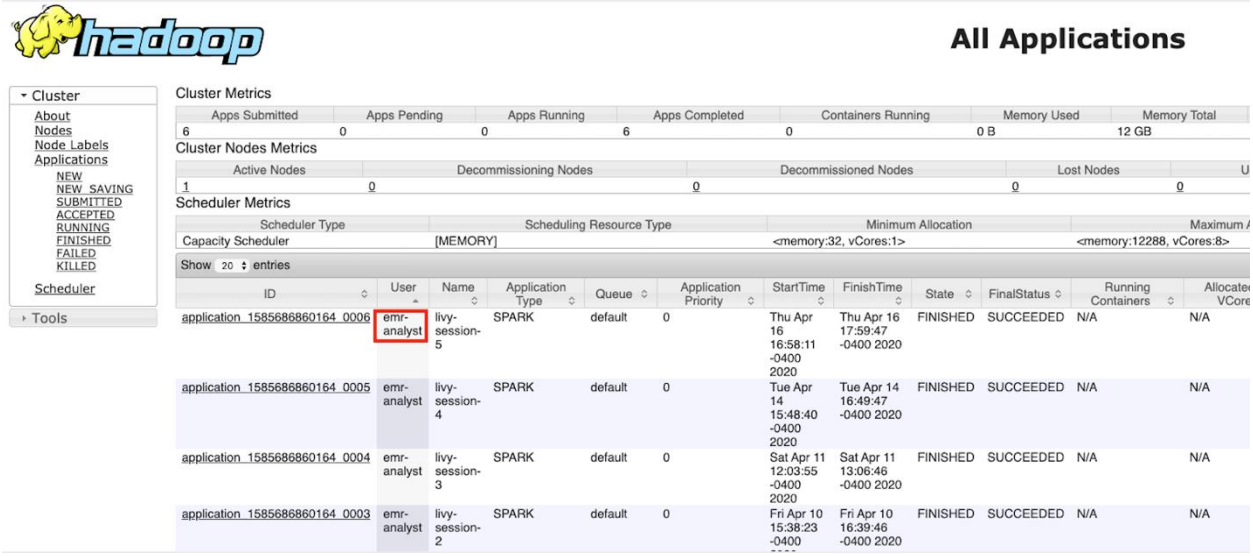


Figure 33: YARN job submitted under authenticated user

For more information about setting up a trust relationship between SAML providers and IAM, see Configure Third-Party Providers for SAML. For more information about IAM Roles needed for this integration, see IAM Roles for Lake Formation, and on how to launch an Amazon EMR cluster, see Launch an Amazon EMR cluster with Lake Formation.

Data Isolation

After the users are authenticated, you must consider what data assets they are authorized to use. You can choose to implement authorization on Amazon EMR at the storage layer or server layer. By default, the policy attached to the EC2 role on your cluster determines the data that can be accessed in Amazon S3. With EMR File Systems (EMRFS) authorization, you can specify the IAM role to assume when a user or group uses EMRFS to access Amazon S3. Choosing the IAM role for each user or group enables fine-grained access control for S3 on multiuser Amazon EMR clusters.

Note: EMRFS doesn't prevent users that run Spark applications or users that access the EMR cluster via SSH from bypassing EMRFS and assuming different IAM roles. For more information, see AWS Lake Formation.

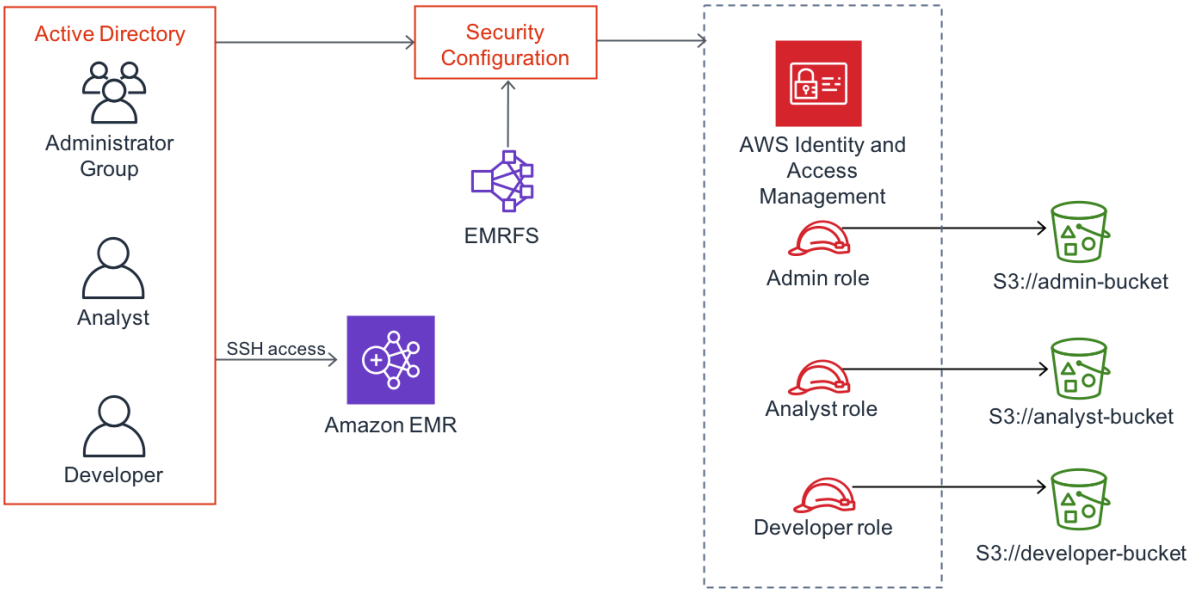


Figure 34: Authorization at storage and server layers

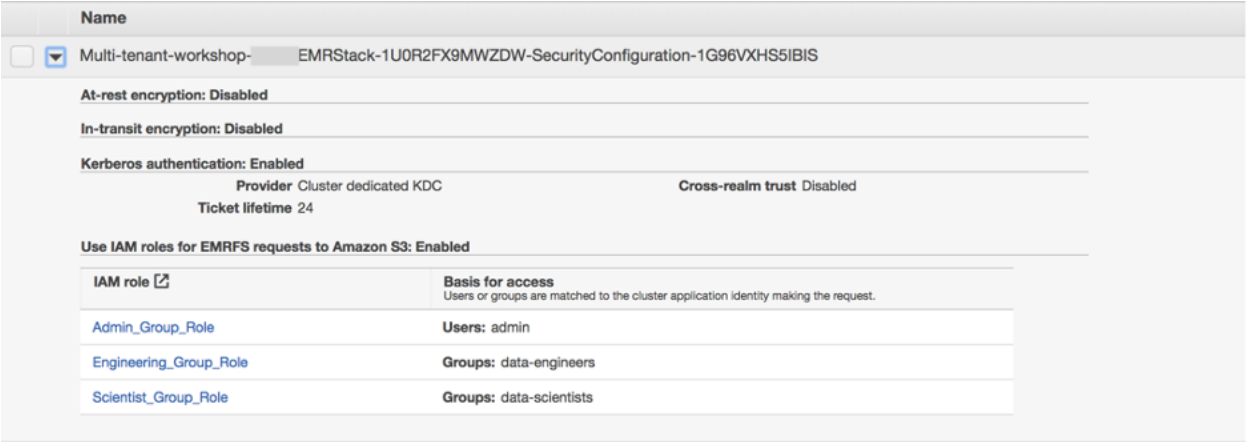


Figure 35: Fine-grained access control with EMRFS authorization

Fine-Grained Access control with AWS Lake Formation:

Integrating Amazon EMR cluster with AWS Lake Formation provides fine-grained, column-level access to databases and tables in the AWS Glue Data Catalog. After the user authenticates through federated single sign-on to EMR Notebooks or Apache Zeppelin, the user can use Spark SQL to access only the

table and columns authorized in shared mode/multi-tenant cluster. The AWS Lake Formation admin can define and manage permissions access to databases, tables, and columns in AWS Glue Data Catalog in AWS Lake Formation. For more information, see AWS Lake Formation.

Resource Isolation

On Amazon EMR, you can use different YARN queues to submit jobs. Each of the YARN queues may have a different resource capacity and be associated with specific users and groups on the Amazon EMR cluster. The YARN Resource Manager UI shows a list of available queues.

> **Note:** YARN queues apply to only applications that run on YARN. For example, YARN queues are not used by Presto applications.

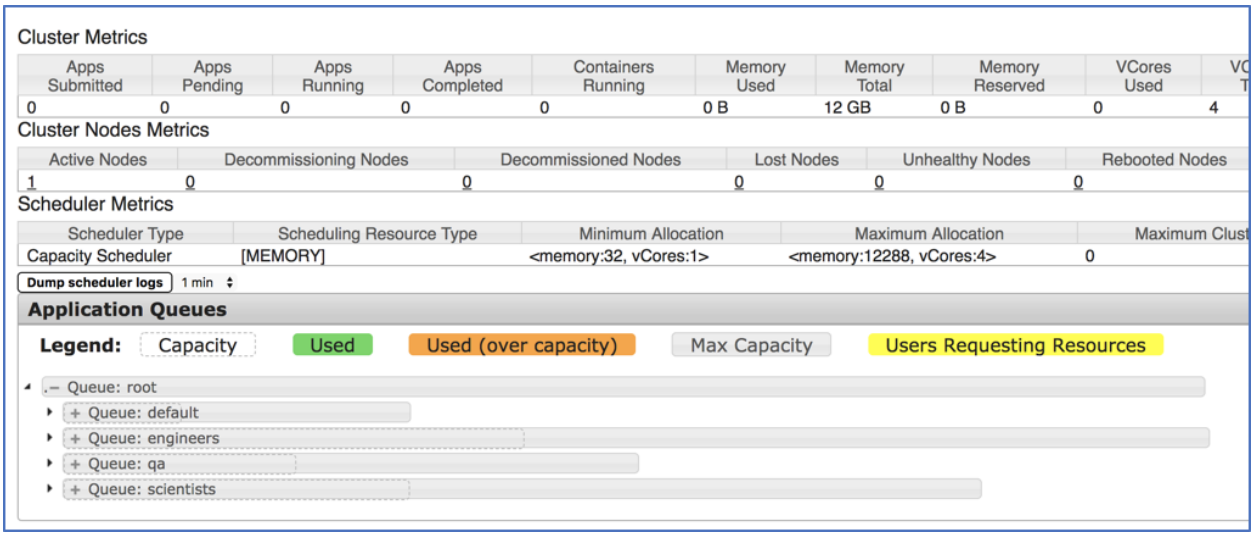


Figure 36: YARN Resource Manager UI

For example, a user **engineer** from the **engineers** group can log in to the EMR primary node and submit jobs to the **engineers** YARN queue:

```
ssh -l engineer <<emr-dns>>
[engineer@ 10-10-1-222 ~] spark-submit --queue engineers --class
org.apache.spark.examples.SparkPi --master yarn
/usr/lib/spark/examples/jars/spark-examples.jar
```

```
[engineer@ip-10-0-1-222 ~]$ spark-submit --queue engineers --class org.apache.spark.examples.SparkPi --master yarn /usr/lib/spark/examples/jars/spark-examples.jar
18/11/13 19:03:11 INFO SparkContext: Running Spark version 2.3.1
18/11/13 19:03:11 INFO SparkContext: Submitted application: Spark Pi
18/11/13 19:03:11 INFO SecurityManager: Changing view acls to: engineer
18/11/13 19:03:11 INFO SecurityManager: Changing modify acls to: engineer
18/11/13 19:03:11 INFO SecurityManager: Changing view acls groups to:
18/11/13 19:03:11 INFO SecurityManager: Changing modify acls groups to:
18/11/13 19:03:11 INFO SecurityManager: SecurityManager: authentication disabled; ui acls disabled; users  with view permissions: Set(engineer); groups with view permissions: Set();
 users  with modify permissions: Set(engineer); groups with modify permissions: Set()
```

Figure 37: Example code on Amazon EMR

In this previous example code, a user `engineer` is submitting a Spark job and passing a parameter – `queue` to reflect which queue it should use to run that Spark job. The YARN ResourceManager UI shows the same job being performed.

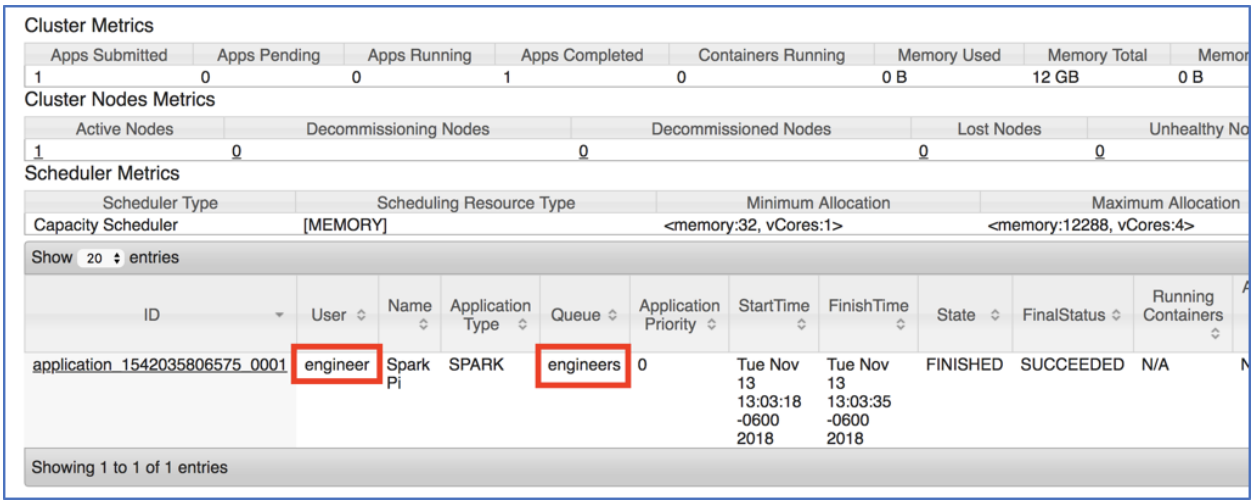


Figure 38: User engineer submitting Spark job in YARN

Extract, Transform, Load (ETL) on Amazon EMR

Orchestration on Amazon EMR

Orchestration frameworks are heavily used in the Apache Hadoop ecosystem to integrate and monitor multiple applications centrally. They manage complex interdependent jobs, maintain each of the applications' states, and execute based on a pre-defined pattern. There are a number of orchestration engines that are available for Hadoop, among those Apache Oozie is a widely used workflow scheduler system. You can also use AWS Step Functions, AWS Glue or Apache Airflow to orchestrate and schedule multiple applications. When migrating workloads from an on-premises Hadoop cluster to Amazon EMR, in addition to the applications that are running on the Hadoop cluster, the orchestration tool must be migrated so that it can effectively orchestrate applications in the cloud. This section discusses the steps to migrate an orchestration application from an on-premises Hadoop cluster, provides other orchestration options available for Amazon EMR, and details recommended best practices for migration.

Migrating Apache Oozie to Amazon EMR

Apache Oozie is a Java web application that manages and schedules Apache Hadoop jobs. Oozie combines multiple jobs and creates a directed acyclic graph (DAG) of actions that can be scheduled based on time or events. Oozie uses a database that stores each jobs' metadata. By default, Oozie is configured to use the embedded Apache Derby database. However, for better reliability and availability in production workloads, make sure to use an external database, such as MySQL, PostgreSQL, or Oracle. Oozie uses XML to define its workflows and coordinators.

Oozie is included with Amazon EMR release version 5.0.0 and later. You can select Oozie in Software Configuration section of the Amazon EMR console.

You can also select Oozie by configurating options through the AWS CLI:

```
aws emr create-cluster --release-label emr-5.17.0 --instance-type
m4.large --instance-count 2 --applications Name=Hadoop Name=Hive
Name=Oozie --name "TestCluster"
```

When migrating Apache Oozie from an on-premises Hadoop cluster to Amazon EMR, first migrate the Oozie database (optional), then migrate the Oozie jobs.

Migrate Oozie Database

These steps are only required if you want to migrate your Oozie job history from the on-premises cluster to Amazon EMR. In this scenario, some of the Oozie database links may break because they refer to the old cluster's host addresses. Therefore, we recommend that you start with an empty Oozie database when migrating to Apache Oozie on Amazon EMR.

Apache Oozie comes with a dump and load utility that you can use to dump an existing Oozie database to a local file, then install and configure the empty database in which to load your Oozie data. And finally, import the database from the backup file.

1. Log in to the host where the existing Oozie server is running and execute this command.

   Note: In Amazon EMR, the `oozie-setup.sh` file is located in this file path: `/usr/lib/oozie/bin/`

```
./oozie-setup.sh export /folder/oozie_db.zip
```

2. Using AWS CLI, upload the `oozie_db.zip` file into Amazon S3:

```
aws s3 cp oozie_db.zip s3://migration/oozie/oozie_db.zip
```

3. Log in to Amazon EMR master node and download the `oozie_db.zip` file from Amazon S3.

```
ssh -i <<key>> hadoop@<<amazon-emr-master-node>>
aws s3 cp s3://migration/oozie/oozie_db.zip oozie_db.zip
```

4. Change the `oozie-site.xml` file to point to a new target database. Here is the default configuration related to Oozie database settings:

```
oozie.db.schema.name=oozie
oozie.service.JPAService.create.db.schema=false
oozie.service.JPAService.validate.db.connection=false
oozie.service.JPAService.jdbc.driver=org.apache.derby.jdbc.EmbeddedDriver
oozie.service.JPAService.jdbc.url=jdbc:derby:${oozie.data.dir}/${oozie.db.schema.name}-db;create=true
oozie.service.JPAService.jdbc.username=sa
oozie.service.JPAService.jdbc.password=
oozie.service.JPAService.pool.max.active.conn=10
```

   For example, if you are planning to use MySQL on Amazon RDS, create the Amazon RDS instance and then update the `oozie-site.xml` file to reflect the RDS configuration.

```
<property>
    <name>oozie.service.JPAService.jdbc.driver</name>
    <value>com.mysql.jdbc.Driver</value>
</property>
<property>
    <name>oozie.service.JPAService.jdbc.url</name>
    <value>jdbc:mysql://<<rds-host>>:3306/oozie</value>
    </property>
<property>
    <name>oozie.service.JPAService.jdbc.username</name>
    <value>mysql-username</value>
</property>
```

```
<property>
    <name>oozie.service.JPAService.jdbc.password</name>
    <value>mysql-password</value>
</property>
```

5. Create the Oozie database in the new RDS instance and grant the required privileges to the Oozie user:

```
$ mysql -u root -p
Enter password:

mysql> create database oozie default character set utf8;
Query OK, 1 row affected (0.00 sec)

mysql> grant all privileges on oozie.* to 'oozie'@'localhost'
identified by 'oozie';
Query OK, 0 rows affected (0.00 sec)

mysql> grant all privileges on oozie.* to 'oozie'@'%' identified by
'oozie';
Query OK, 0 rows affected (0.00 sec)
```

6. Load the previous `oozie_db.zip` database dump to this new database on AWS:

```
./oozie-setup.sh import oozie_db.zip
```

After importing, the CLI shows how many database rows you have imported and their respected table names.

```
4 row(s) imported to table WF_JOBS.
Persisting last batch. [batch.size=12]
12 row(s) imported to table WF_ACTIONS.
0 row(s) imported to table COORD_JOBS.
0 row(s) imported to table COORD_ACTIONS.
0 row(s) imported to table BUNDLE_JOBS.
0 row(s) imported to table BUNDLE_ACTIONS.
0 row(s) imported to table SLA_REGISTRATION.
0 row(s) imported to table SLA_SUMMARY.
Cleanup not necessary, no entities skipped.
3239 [main] INFO org.apache.oozie.service.Services - Shutdown
```

Figure 39: Successful database import

7. Restart the Oozie server to reflect the new configuration and database:

```
sudo restart oozie
```

> **Note:** Depending on the database, the driver's jar file needs to be placed in the Oozie classpath on Amazon EMR, the location is: `/var/lib/oozie`.

Migrating Oozie Jobs

Migrating Oozie jobs from an on-premises Hadoop cluster to Amazon EMR is straight-forward. If you have not migrated an Oozie database to Amazon EMR, then we recommend you just move all Oozie job-related files to Amazon EMR. Moving job-related files includes the following files:

- job.properties
- workflow.xml
- coordinator.xml
- Bundle files
- external parameter files
- any dependent file

To move these files quickly, take these steps:

1. Compress all of the Oozie related files into a single archive file.
2. Upload that archive file to Amazon S3.
3. Download that file to an Amazon EMR primary node.
4. Extract the compressed file to appropriate folders.
5. Modify the individual files to reflect the Amazon EMR cluster settings.
6. Resubmit those Oozie jobs on Amazon EMR.

Considerations

Consider these issues when migrating the Oozie database and jobs.

- The Oozie native web interface is not supported on Amazon EMR. To use a frontend interface for Oozie, use the Hue application running on Amazon EMR. For more information, see Hue.
- Like Amazon EMR, by default, Oozie is configured to use the embedded Derby database. We recommend that you use an Amazon RDS instance to host an Oozie database.

- You can use Amazon S3 to store workflow XML files. To do so, you must place EMRFS library files in the Oozie classpath. Follow these steps:

  a. Open a command-line and use this command to locate the `emrfs-hadoop-assembly` file in the Amazon EMR primary node.

```
[hadoop@ip-10-0-30-172 spark]$ locate emrfs-hadoop-assembly
/usr/share/aws/emr/emrfs/lib/emrfs-hadoop-assembly-2.26.0.jar
```

  b. Copy the emrfs-hadoop-assembly file to the `/usr/lib/oozie/lib` directory.

```
[hadoop@ip-10-0-30-172 spark]$ sudo cp
/usr/share/aws/emr/emrfs/lib/emrfs-hadoop-assembly-2.26.0.jar
/usr/lib/oozie/lib
```

  c. Restart the Oozie server.

```
[hadoop@ip-10-0-30-172 spark]$ sudo restart oozie
```

The following is a sample `workflow.xml` file that executes SparkPI using Oozie's Spark action. In this sample, `spark-examples-jar` and `workflow.xml` are stored in an Amazon S3 bucket (`s3://tm-app-demos/oozie`).

```
<workflow-app xmlns='uri:oozie:workflow:1.0' name='SparkPi-S3'>
<start to='spark-node' />

<action name='spark-node'>
<spark xmlns="uri:oozie:spark-action:1.0">
<resource-manager>${resourceManager}</resource-manager>
<name-node>${nameNode}</name-node>
<master>${master}</master>
<name>SparkPi</name>
<class>org.apache.spark.examples.SparkPi</class>
<jar>s3://tm-app-demos/oozie/lib/spark-examples.jar</jar>
</spark>
<ok to="end" />
<error to="fail" />
</action>

<kill name="fail">
<message>Workflow failed, error
message[${wf:errorMessage(wf:lastErrorNode())}]
</message>
</kill>
```

```
<end name='end' />
</workflow-app>
```

This sample is the corresponding job.properties file. See *Figure 40* for the Amazon S3 bucket that is created for the Oozie files.

```
nameNode=hdfs://ip-10-0-30-172.ec2.internal:8020
resourceManager=ip-10-0-30-172.ec2.internal:8032
master=yarn-cluster
queueName=default
examplesRoot=examples
oozie.use.system.libpath=true
#oozie.wf.application.path=${nameNode}/user/${user.name}/oozie/spar
k
oozie.wf.application.path=s3://tm-app-demos/oozie
```

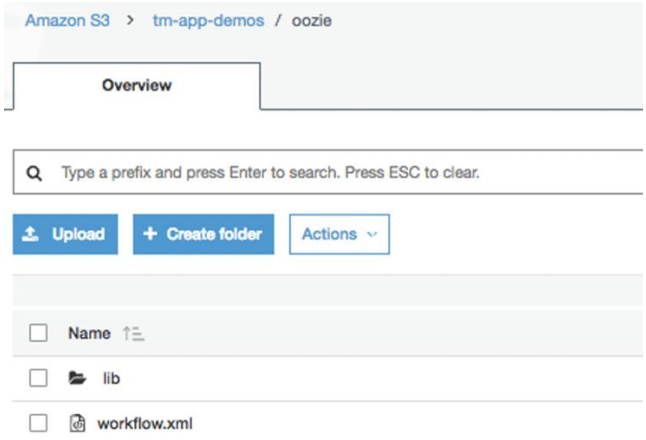


Figure 40: Amazon S3 bucket for Oozie related files

AWS Services for Orchestration

AWS services can be used to create orchestration for Hadoop-based jobs. The following services are some of the popular orchestration options on AWS.

AWS Step Functions

AWS Step Functions lets you coordinate multiple AWS services into serverless workflows so you can build and update applications quickly. Using Step Functions, you can design and run workflows that stitch together services. Workflows are made up of a series of steps, with the output of one step acting as input into the next. You can monitor each step of execution as it happens, which means you can identify and fix problems quickly. Step Functions automatically triggers and tracks each step, and retries when there are errors, so your application executes in order and as expected.

You can connect to Amazon EMR from AWS Step Functions to build data processing and analytics workflows. With minimal code, you can orchestrate Amazon EMR provisioning using Step Functions. The integration between Amazon EMR and Step Functions depends on EMR Service Integration APIs. Using those APIs, a Step Functions state machine can:

- Create or terminate your Amazon EMR cluster. You can reuse the same cluster in your workflow or can create the cluster on-demand based on your workflow.
- Add or cancel an EMR step. Each step is a unit of work that contains instructions to manipulate data for processing by software installed on the cluster. By using this you can submit Apache Spark, Apache Hive, or Presto jobs to an Amazon EMR cluster. You can also create dependencies between multiple steps or can design them to run in parallel.
- Modify the size of an EMR cluster. This allows you to scale your EMR programmatically depending on the requirements of each step of your workflow.

The following image is an example of how AWS Step Functions orchestrates multiple Apache Spark jobs.

Figure 41: Multiple Apache Spark jobs orchestrated with AWS Step Functions

For more information on how to integrate AWS Step Functions to create orchestration for Hadoop-based jobs, see these blog posts:

- Using Step Functions to Orchestrate Amazon EMR Workloads
- Orchestrate Apache Spark applications using AWS Step Functions and Apache Livy

Other Orchestration Options

Some open source applications for orchestration have gained popularity due to community adoption and a rich feature set. This section covers Apache Airflow and Luigi and how these applications can be used on AWS to create orchestration for Hadoop-based jobs.

Apache Airflow

Airflow is an open-sourced task scheduler that helps manage ETL tasks. Apache Airflow workflows can be scheduled and managed from one central location. With Airflow's Configuration as Code approach, automating the generation of workflows, ETL tasks, and dependencies is easy. It helps developers shift their focus from building and debugging data pipelines to focusing on the business problems.

Apache Airflow can be installed on an Amazon EC2 instance or on an Amazon EMR primary node through bootstrap. It comes with a variety of connectors that help to integrate it with different AWS services.

For more information on how Airflow can be used to build orchestration pipeline and how it can be integrated to run jobs on Amazon EMR, check the following posts on the AWS Big Data Blog:

- Build a Concurrent Data Orchestration Pipeline Using Amazon EMR and Apache Livy
- Orchestrate big data workflows with Apache Airflow, Genie, and Amazon EMR

Luigi

Luigi is another open-sourced application you can use to build a complex pipeline of batch jobs. It handles scheduling, dependency resolution, workflow management and includes command line tool integration.

Best Practices for Orchestration

There are some points to be consider when building robust and fault-tolerant orchestration for Apache Hadoop-based jobs. Like Hadoop, an orchestration tool should be scalable so that it can handle a massive number of jobs and can scale proportionally with Hadoop scaling. Here are some of the best practices to consider when creating orchestration for Hadoop jobs:

- Most of the orchestration applications use a default, embedded database to store job metadata. For production workloads, we recommend that you use a separate database for better performance and availability.

- When possible, use serverless or managed orchestration services to reduce ongoing manual involvement.
- Integrate with notification services, such as Amazon SNS and Amazon CloudWatch, so that appropriate parties are immediately notified upon failure and can be involved proactively.
- Make sure that the orchestration application can handle both asynchronous and synchronous job and task execution for better performance and reduced overhead.
- The orchestration application should monitor job execution and management so that developers can monitor everything centrally.
- The orchestration application should be able to handle failure gracefully.
- If you use Apache Airflow, make sure to use cluster-mode configuration for production workloads.

Use the following table to determine the most appropriate orchestration application for your use case.

Table 5: Orchestration applications

Factors/Use Cases	AWS Step Functions	Apache Oozie	Apache Airflow
Serverless	Yes	No	No
Spark-based jobs	Yes	Yes	Yes
Rich UI & troubleshooting tools	No	No	Yes
Integration with other monitoring tools	Yes	No	Yes
Interacting with AWS Services	Extensive	Very Limited	Fair
Administrative responsibilities	Serverless	Light	Huge
Hybrid environment – AWS and non-AWS services	Only in the cloud	Only for Hadoop jobs	Broad coverage
Cost	$0.025 per 1,000 state transitions	EMR usage	EC2 usage

Migrating Apache Spark

Apache Spark applications are a common workload on Amazon EMR. Because Amazon EMR clusters can be configured to use specific instance types and can easily scale out to a large number of workloads, Apache Spark is effective for optimizing compute resources for both performance and cost.

Use Cases for Migrating Apache Spark

There are several use cases to consider when migrating Apache Spark applications to Amazon EMR. In many cases, the existing environment is one large cluster that has a specific amount of resources dedicated to processing Spark jobs. With Amazon EMR, you can continue to use one large shared cluster, or you can use On-Demand Instance clusters to isolate resources on a per-job basis. The On-Demand Instance approach allows you to take advantage of different instance types in addition to ensuring that Spark is fully using the resources of each cluster. However, this approach does require more planning and automation around the creation of Amazon EMR clusters before running a Spark job.

Shared Cluster

In a shared cluster, be aware of how many concurrent jobs you expect to run at any given time. By default, EMR configures executors to use the maximum number of resources possible on each node through the usage of the maximizeResourceAllocation property. On a shared cluster, you may need to manually configure Spark cores, memory, and executors. For details, see Best practices for successfully managing memory for Apache Spark applications on Amazon EMR on the AWS Big Data Blog. The shared cluster is appropriate for interactive use cases, such as when you are using Jupyter notebooks.

When using a shared cluster, we recommend that you use the dynamic allocation setting in Amazon EMR to both automatically calculate the default executor size and to allow resources to be given back to the cluster if they are no longer used. Dynamic allocation is enabled by default on Amazon EMR.

Dedicated Clusters per Job

Using a separate cluster per each Spark job is beneficial for scheduled Spark jobs. This approach helps isolate the Spark job to prevent resource contention, allows for optimization of the job depending on if it's a CPU-, GPU-, or memory-intensive workload, and ensures that you only pay for the resources you use during the duration of the job. The Amazon EMR `maximizeResourceAllocation` setting helps ensure that the entire cluster's resources are dedicated to the job. For more information, see Using maximizeResourceAllocation in the Amazon EMR Release Guide.

Amazon EMR Spark Runtime

Amazon EMR runtime for Apache Spark is a performance-optimized runtime environment that is active by default on Amazon EMR clusters starting in EMR release 5.28.0. EMR runtime for Spark is up to 32 times faster than EMR 5.16, with 100% API compatibility with open-source Spark.

To measure the impact of these improvements, we used TPC-DS benchmark queries with 3-TB scale running on a 6-node c4.8xlarge EMR cluster with data in Amazon S3. We measured performance improvements as the geometric mean of improvement in total query processing time, and total query processing time across all queries. These improvements means that your workloads run faster, saving you compute costs without making any changes to your applications.

Optimize Cost with Amazon EC2 Spot Instances

A common way to decrease cost with Spark jobs on EMR is by using EC2 Spot Instances. When using EMR release version 5.9.0 or later, Spark on EMR includes a set of features to help ensure that Spark gracefully handles node termination in case of a manual resize or an automatic scaling policy request.[3]) Amazon EMR 5.11.0 includes a `spark.decommissioning.timeout.threshold` that further improves Spark resiliency when using Spot Instances. Therefore, if you use Spot Instances, make sure that you are using Amazon EMR release version 5.11.0 or later.

Use Instance Fleets

Instance fleets are a feature of Amazon EMR that allows you to specify target capacity based on a specific set of units. These units could represent cores, memory, or any arbitrary reference. Instance fleets are useful if you know that your Spark job requires a certain amount of resources, and you want to mix and match capacity across both EC2 instance type and Availability Zone. With instance fleets, you select a VPC network and a set of EC2 subnets, and the feature searches Availability Zones in the subnets you selected until it finds the desired capacity.

Unfortunately, Instance Fleets do not support multiple core or task groups or allow for automatic scaling. If you have a dynamic workload that requires either of those features, you must use Instance Groups. For more information on configuring your cluster, see Cluster Configuration Guidelines and Best Practices in the Amazon EMR Management Guide.

Apache Spark File Write Performance

In Amazon EMR 5.14.0, the default `FileOutputCommitter` algorithm has been updated to use version 2 instead of version 1. This update reduces the number of renames, which improves application performance. Any Spark applications being migrated to EMR should use the most recent available Amazon EMR version to take advantage of this update and other performance improvements. In addition, Amazon EMR version 5.20.0 includes an S3-optimized committer that is enabled by default. The EMRFS S3-optimized committer is an alternative OutputCommitter implementation that is optimized for writing files to Amazon S3 when using EMRFS.

Amazon S3 Select with Apache Spark

In certain scenarios, using S3 Select with Spark can result in both increased performance and decreased amount of data transferred between Amazon EMR and Amazon S3. S3 Select allows applications to

retrieve only a subset of data from an object. As of EMR 5.17.0, S3 Select is supported with CSV and JSON files. If your query filters out more than half of the original dataset and your network connection between Amazon S3 and EMR has good transfer speed, S3 Select may be suitable for your application.

EMRFS Consistent View

When chaining Spark jobs that write data to S3, ensure that the data written by one job is the same data read by subsequent jobs. Due to the S3 Data Consistency Model, object metadata may not always be available if you immediately list objects after a large set of writes. With consistent view enabled, EMRFS maintains a metadata store of objects it expects to be in Amazon S3 and returns the set of objects listed in that metadata store for subsequent jobs.

> **Note:** Consistent View is intended for a set of chained jobs or applications that control all reads and writes to S3, such as an Apache HBase on S3 EMR deployment. It is not intended to be used for a globally consistent view of all objects in your S3 bucket.

AWS Glue Data Catalog

For Amazon EMR 5.8.0 and later, Spark supports using the AWS Glue Data Catalog as the metastore for Spark SQL. If you plan to migrate to Spark on EMR, first determine if you can migrate your existing metastore to AWS Glue. Using AWS Glue Data Catalog has the benefit of not just being a managed metadata catalog, but it also integrates with a number of different AWS products, including Apache Presto and Apache Hive on EMR, Amazon Athena, Amazon Redshift Spectrum, Amazon SageMaker, and Amazon Kinesis Data Firehose. In combination with AWS Glue crawlers, the Data Catalog can generate schema and partitions and provide for fine-grained access control to databases and tables.

Troubleshooting Apache Spark jobs on an Amazon EMR Cluster

To debug common issues, view the history and log files of these applications:

- For Spark Web UIs, access the Spark HistoryServer UI port number at 18080 of the EMR cluster's master node. For more information, see Accessing the Spark Web UI
- For YARN applications, including Spark jobs, access the **Application history** tab in the Amazon EMR console. Up to seven days of application history is retained, including details on task and stage completion for Spark. For more information, see View Application History.
- By default, Amazon EMR clusters launched via the console automatically archive log files to Amazon S3. You can find raw container logs in Amazon S3 and view them while the cluster is active and after it has been terminated. For more information, see View Log Files.

Migrating Apache Hive

Apache Hive is one of the popular applications used by Amazon EMR customers as a data warehouse. As with any migration, you have several considerations to make.

Hive Metastore

By default, Amazon EMR clusters are configured to use a local instance of MySQL as the Hive metastore. To allow for the most effective use of Amazon EMR, you should use a shared Hive metastore, such as Amazon RDS, Amazon Aurora, or AWS Glue Data Catalog.[4] If you require a persistent metastore, or if you have a metastore shared by different clusters, services, applications, or AWS accounts, we recommend that you use an AWS Glue Data Catalog as a metastore for Hive. For more information, see Configuring an External Metastore for Hive.

Upgrading

Hive upgrades couple with Hive metastore updates. The Hive metadata database should be backed up and isolated from production instances because Hive upgrades may change the Hive schema, which may cause compatibility issues and problems in production. You can perform upgrades using the `–upgradeSchema` command in the Hive Schema Tool. You can also use this tool to upgrade the schema from an older version to the current version.

Hive Execution Engine

Apache TEZ is the supported and default execution engine for Hive clusters on EMR. In most cases, Tez provides improved performance. However, if you are migrating from the older version of Hive that used the MapReduce execution (MR) engine, certain jobs may require changes.

Hive 2.3.0 added support for Apache Spark as an execution engine, but this setup is not supported on EMR without changes to the underlying Hive jars in Spark. This is not a supported configuration on EMR.

Tez Container Size

If you have large input files that cannot be split, or if the map portion of a job exceeds the default memory limits of a container, you will require a larger Tez container size. On Amazon EMR, the default container setting is `hive.tez.container.size` and is set to **-1**. This means that the value of `mapreduce.map.memory.mb` is used for the memory limit. The default values of `mapreduce.map.memory.mb` depend on the specific instance type selected for your EMR cluster. For this setting and other default values, see Task Configuration.

The desired value of the Tez container size depends upon the specifics of your job. It must be at least as the same value as the `mapreduce.map.memory.mb` setting.

If the Tez container runs out of memory, the following error message appears:

```
Container
[pid=1234,containerID=container_1111222233334444_0007_02_000001] is
running beyond physical memory limits. Current usage: 1.0 GB of 1
GB physical memory used; 1.9 GB of 5 GB virtual memory used.
Killing container.
```

To increase the memory, set the `hive.tez.container.size` to a value greater than what is required for the job. The memory value required for the job can be found in the error message. In addition to container size, increase the Tez Java Heap size. In general, the Tez Java Heap size should be 80% of the Tez Container size. You can adjust the value of the Tez Java Heap size with the setting `hive.tez.java.opts`.

HDFS vs S3 Considerations

A benefit of EMR is the ability to separate storage and compute requirements through the use of Amazon S3 as your primary data store. This approach allows you to save costs compared to HDFS by scaling your storage and compute needs up or down independently. Amazon S3 provides infinite scalability, high durability and availability, and additional functionality such as data encryption and lifecycle management. That said, Hadoop was designed with the expectation that the underlying filesystem would support atomic renames and be consistent. There are several options to consider if you require immediate list and read-after-write consistency as part of your workflow.

EMRFS Consistent View

EMRFS consistent view is an optional feature that allows EMR clusters to check for list and read-after-write consistency for Amazon S3 objects written by or synced with EMRFS. It is ideal for working with chained Apache MapReduce jobs writing to S3 or specific applications on EMR, such as Apache HBase, that control all read and write operations to S3.

Hive Tez Merge files

Merge files performs a number of operations that invalidate S3 read-after-write consistency and can result in missing data. Therefore, we recommend that you do not use the `hive.merge.tezfiles` without enabling EMRFS consistent view. An alternative would also be performing any `INSERT OVERWRITE` or `ALTER CONCATENATE` statements on transient HDFS-backed EMR clusters and copying the results back to S3 using `s3-dist-cp`.

Other ways that EMR customers have solved large-scale consistency issues include implementing a custom manifest file approach to their jobs instead of using S3 list operations to retrieve data, or by building their own metadata stores, such as Netflix Iceberg.

Hive Blobstore Optimization

Hive blobstore optimizations are intended to increase the performance of intermediate MapReduce jobs on Hive. However, when performing simple Hive queries with S3-backed tables, such as `INSERT`

`OVERWRITE` or `ALTER TABLE CONCATENATE`, this setting can sometimes result in increased execution times and missing data. This issue is caused by implementing that feature on queries that are not multi-staged MapReduce jobs. This scenario results in Hive using Amazon S3 as the scratch directory during the job. As a result, the number of renames on S3 increases as the job progresses through writing scratch data, copying to another S3 temporary location, and finally copying to the final S3 location.

If you disable this setting, the scratch directory for the job is relocated to HDFS. If you prefer this scenario, make sure to allocate enough space on the EMR cluster to accommodate this change. By default, the `distcp` job that occurs at the end of the process is limited to a maximum of 20 mappers. If you find this job is taking too long, particularly if you are processing terabytes of data, you can manually set the number of max mappers using the following code in your Hive job:[5]

```
SET distcp.options.m=500
```

Job Throughput and Scheduling

By default, EMR uses the Hadoop CapacityScheduler for allocating resources in the cluster. The CapacityScheduler allows large cluster sharing and provides capacity guarantees for each organization. The CapacityScheduler supports queues that can allow an organization to grant capacity in a multitenant cluster. You can configure the CapacityScheduler by modifying the `capacity-scheduler` classification during EMR cluster creation.

In some cases, the FairScheduler may be more desirable for clusters where it is acceptable for a job to consume unused resources. In FairScheduler, resources are shared between queues.

Configure FairScheduler

You can configure FairScheduler in a couple ways when creating an EMR cluster.

Modify the yarn-config.json file

1. Use the following configuration below in the `yarn-config.json` file.

```
[
  {
    "Classification": "yarn-site",
    "Properties": {
      "yarn.resourcemanager.scheduler.class":
"org.apache.hadoop.yarn.server.resourcemanager.scheduler.fair.FairS
cheduler"
    }
  }
]
```

2. Modify the AWS CLI command to match the following code example and use this command to start a cluster with the config file created above.

```
aws emr create-cluster \
        --applications Name=Spark Name=Ganglia \
        --ec2-attributes "${EC2_PROPERTIES}" \
        --service-role EMR_DefaultRole \
        --release-label emr-5.20.0 \
        --log-uri ${S3_LOGS} \
        --enable-debugging \
        --name ${CLUSTER_NAME} \
        --region us-east-1 \
        --instance-groups \
        --configurations file://yarn-config.json
InstanceGroupType=MASTER,InstanceCount=1,InstanceType=m3.xlarge
InstanceGroupType=CORE,InstanceCount=4,InstanceType=m3.xlarge)
```

Edit software settings in the console

1. Sign in to the AWS Management Console and open the Amazon EMR console at https://console.aws.amazon.com/elasticmapreduce/.
2. Choose **Create cluster, Go to advanced options**.
3. Choose **Spark**.
4. Under **Edit software settings**, leave **Enter configuration** selected and enter the following configuration:

```
classification=yarn-
site,properties=[yarn.resourcemanager.scheduler.class=org.apache.ha
doop.yarn.server.resourcemanager.scheduler.fair.FairScheduler]
```

5. Choose **Create cluster**.

Maintaining a Highly Available Hive-Based Cluster

Hive clusters are often long-running due to the nature of ad hoc queries that can come in at any time. There are a couple approaches to maintaining a highly available Hive-based cluster.

Warm Failover

In a warm failover scenario, a secondary, smaller cluster is kept running in addition to the primary cluster. If a failure occurs, clients can be redirected to the new cluster, either manually or by updating an entry in Amazon Route 53. You can configure the secondary cluster with a small number of nodes, and then if it becomes the primary cluster, use automatic scaling to increase the number of nodes.

Multi-Cluster Configuration
Because all data is stored on S3, all clients do not need to go through the same cluster. You can configure multiple clusters with a load balancer or expose a job submission framework to consumers of the environment. These clusters can be shared among all clients or shared on a per-team basis depending on internal requirements. One of the benefits of this approach is that in the event of a cluster failure, the impact to you users are limited to just those queries executing on the single cluster that fails. In addition, you can configure automatic scaling so that each cluster scales independently of each other. If the clusters are segmented on a per-team basis, this approach ensures that any one team's jobs don't impact the performance of another team's jobs.

However, using multiple clusters means using multiple master nodes, one each for a cluster. Therefore, you need additional EC2 instances that you wouldn't have to pay for if you used only a single cluster. However, with the EC2 instance pricing model of pay-per-second with a one-minute minimum, in the case of multiple clusters, you can save costs by choosing to activate only the cluster needed to perform the tasks rather than running one single cluster all of the time. You can configure the logic for this setup inside an AWS Lambda function that calls the activation on the pipeline. Then, you can start up or take down a cluster without impacting another cluster's activities.

Transient Design
Transient clusters can mitigate the cost and operational requirements of having a single, long-running cluster. This approach is useful if you have predictable short-lived jobs, but may not be appropriate if you have consumers that are constantly querying data on an ongoing basis.

Frequently Asked Questions

How do I implement transactions and compactions on Amazon EMR?
Transactions and compactions on Amazon EMR can be implemented by using Apache Hudi starting in release 5.28.0. Apache Hudi is an open-source data management framework used to simplify incremental data processing and data pipeline development. For more information, see the Incremental Data Processing section in this guide.

How to I troubleshoot loading data from Amazon S3?
A common mistake is using Amazon S3 like a typical file system. There are certain differences that you must consider if you're moving from HDFS to Amazon S3. For details, see Are you experiencing trouble loading data to or from Amazon S3?.

Does Amazon EMR support Hive LLAP?
Amazon EMR release 6.0.0 and later supports the Live Long and Process (LLAP) functionality for Hive. Hive LLAP uses persistent daemons with intelligent in-memory caching to improve query performance compared to the previous default Tez container execution mode. For more details on using Hive LLAP, see Amazon EMR Hive LLAP

The Hive LLAP daemons are managed and run as a YARN service. Since a YARN service can be considered a long-running YARN application, some of your cluster resources are dedicated to Hive LLAP and cannot be used for other workloads. For more information, see LLAP and YARN Service API.

Amazon EMR Notebooks

Amazon EMR Notebooks provide capability for SQL developers and data engineers to run ad-hoc queries using managed and serverless Jupyter notebooks. Unlike a traditional Jupyter notebook, EMR notebooks are persisted in Amazon S3 separately from the cluster that runs the code. This approach provides the flexibility to detach a notebook from a running cluster and attach the EMR notebook to a different cluster. That is, one notebook is not locked to a single cluster and can be at any time linked to a cluster with different configurations as long as the cluster meets the requirements mentioned here.

Several EMR notebooks can be attached to the same cluster based on the memory constaints of the master node instance type. This feature helps provide a multi-tenant environment for several users to have their own notebooks. You can configure AWS IAM roles so that the notebook of one IAM user cannot be seen or accessed by another user in the same account. EMR Notebooks also provide seamless Git and Bit Bucket integration – users have the ability to link their GitHub or BitBucket repositories to an EMR notebook and check in their code to one or more linked repositories.

For more information, see Using Amazon EMR Notebooks in the Amazon EMR Management Guide,

Comparison of JupyterHub on EMR vs. Amazon EMR Notebooks

The following table provides a comparison of JupyterHub on EMR versus Amazon EMR Notebooks.

Attribute	JupyterHub on EMR	EMR Managed Notebooks
Custom Packages	For Python kernel, custom packages need to be installed manually within the docker container the JupyterHub is running on. For PySpark kernel, from EMR version >= 5.26.0, custom packages can be installed through notebook-scoped libraries on the Spark context itself while running the notebook blocks. On EMR versions < 5.26.0, custom packages need to be installed on all nodes using EMR bootstrap action or at the AMI level.	For Python kernel, to use custom packages, tar installation file should be uploaded manually after which conda offline installation should be performed. For PySpark kernel, from EMR version >= 5.26.0, custom packages can be installed through notebook-scoped libraries on the Spark context itself while running the notebook blocks. On EMR versions < 5.26.0, custom packages need to be installed on all nodes using EMR bootstrap action or at the AMI level.

Attribute	JupyterHub on EMR	EMR Managed Notebooks
Git integration	Currently JupyterLab and Git integration is not supported. But JupyterLab plugin with git extension can be installed through docker commands.	Git and Bit Bucket integration is supported natively and can be used to check out code through JupyterLab.
Notebook storage	Notebooks are stored locally in the cluster. They need to be saved manually and uploaded to S3 or JupyterHub must be configured to automatically save files to S3 during cluster launch.	Notebooks are saved in S3 base location specified during notebook creation.
Flexibility	JupyterHub instance is locked to a single EMR cluster. If the cluster is terminated, the notebooks are deleted as well if not backed up in S3.	Notebook is not locked to one cluster. Since the notebooks are automatically stored in S3, upon cluster termination, it can be re-attached to a different running cluster without losing any data.
Multi-tenancy	All the users have same instance of JupyterHub. So, PAM, LDAP or SAML authentication must be set up to segregate notebooks for each user.	Many notebooks can be attached to a single cluster based on the master node instance type. So, a notebook can be created per-user and the access to another user's notebook can be restricted using IAM policies.
Authentication	LDAP, Kerberos and SAML can be custom configured. Only one of these authentication mechanisms can be applied at a time.	Supports IAM users or SAML authentication through AWS Lake Formation. Kerberized EMR clusters without AWS Lake Formation are not supported.

Migrating Jupyter Notebooks to Amazon EMR Notebooks

To migrate your Jupyter notebooks from your previous installation to EMR Notebooks, export and copy your notebook files to an S3 location used by your EMR notebook. If required, you can convert your Python/PySpark files to notebook format and upload the files to the S3 location.

You can export your Jupyter notebook using one of two methods:

- If you have a small number of notebooks, export the notebooks by manually downloading each notebook (Notebook > File > Download As > Notebook [.ipynb]). Downloaded notebook file(s) are in the <filename>.ipynb file format. To copy these files to S3, you can either use the AWS Management Console or use the AWS S3 CLI command. For example:

```
aws s3 cp SharedNotebook.ipynb s3://MyBucket/MyNotebooksFolder/e-
12A3BCDEFJHIJKLMNO45PQRST/MyNotebook.ipynb
```

- If you have several notebooks, you can copy the notebooks directly from the Juptyer installation to S3. To do so, SSH into the node that holds the Jupyter installation and go to where the notebooks are being saved, for example /mnt/var/lib/jupyter/home/jovyan/ and then copy the files by running the following command.

```
aws s3 cp /mnt/var/lib/jupyter/home/jovyan/
s3://MyBucket/MyNotebooksFolder/e-12A3BCDEFJHIJKLMNO45PQRST/ --
recursive
```

If your Python/PySpark programs are stored in .py format, you must first convert them to .ipynb file format before working with them in an EMR notebook. You can use an open source python package called jupytext that has the ability to convert python files into notebook files.

```
sudo pip install jupytext
```

The following code is an example command that converts a .py file to .ipynb file which can then be imported to EMR notebooks.

```
jupytext --to notebook python_file.py
```

Incremental Data Processing

Apache Hudi is an open-source data management framework used to simplify incremental data processing and data pipeline development by providing record-level insert, update, upsert, and delete capabilities. *Upsert* refers to the ability to insert records into an existing dataset if they do not already exist or to update them if they do. By efficiently managing how data is laid out in Amazon S3, Hudi allows data to be ingested and updated in near real time. Hudi carefully maintains metadata of the actions performed on the dataset to help ensure that the actions are atomic and consistent.

Hudi is integrated with Apache Spark, Apache Hive, and Presto. With Amazon EMR release version 5.28.0 and later, Amazon EMR installs Hudi components by default when Spark, Hive, or Presto are installed. You can use Spark or the Hudi DeltaStreamer utility to create or update Hudi datasets. You can use Hive, Spark, or Presto to query a Hudi dataset written on Amazon S3. Hudi maintains the metadata of the actions performed on the Hudi dataset in index and data files. With the metadata written in files, an application can create existing Hudi dataset by loading the metadata from S3, makes it easy to reuse the same data for a variety of use cases.

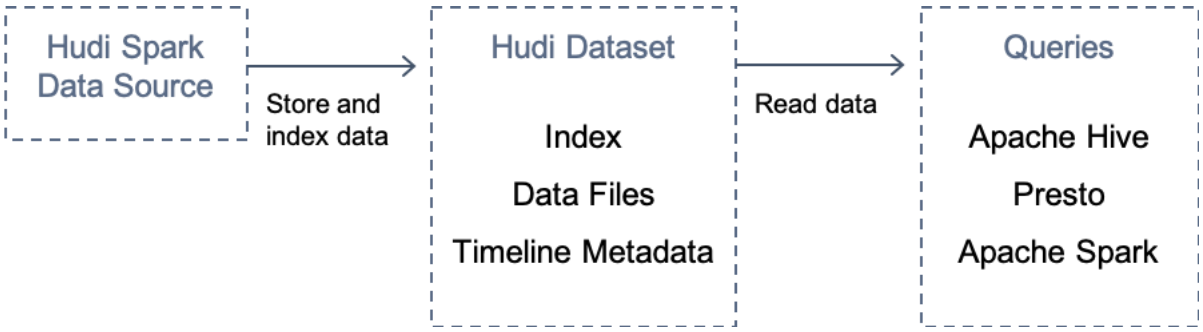


Figure 42: Hudi Architecture

When writing datasets, Hudi supports two table types:

- **Copy on Write:** Stores data in columnar format (Parquet) only. Write operation on this table results in updating the version and rewriting the files using a merge.
- **Merge on Read:** Stores data in both columnar (Parquet) and row (Apache Avro) based formats.

Write operations result in updates stored as delta files. Compactions are run at a scheduled frequency to arrive at new columnar files (synchronously or asynchronously.)

In addition to the ability to perform upserts (updates/inserts), Hudi also provides snapshot isolation for readers (queries), atomic writes of batch of records, incremental pulls and de-duplication of data.

Considerations for using Apache Hudi on Amazon EMR

Assess fit for use case

Consider using Hudi to efficiently solve the problems with incrementally ingesting data into a data lake, enforcing data privacy regulations where consumers might choose to be forgotten/erased (GDPR/CCPA), applying change data capture to data lake, bringing data freshness within minutes to your data marts on S3, and need to provide point in time views of the data in your data lake. It is recommended that you evaluate your use case's data freshness SLA, concurrency needs, query latency and data access patterns to assess if Hudi is the right solution for your workload. Note that Hudi is not a replacement for online transactional processing (OLTP) systems and not ideal where your incoming stream of data is used in an append-only fashion (for example, dataset is primarily not mutable).

Writing Hudi datasets

Hudi provides two ways to write datasets.

- **DeltaStreamer** is a utility included with Hudi that allows you to simplify the process of applying changes to Hudi data sets. DeltaStreamer is a CLI tool, that can operate against three sources – Amazon S3, Apache Kafka, and Apache Hive incremental pull. (*Incremental pull* refers to the ability to pull only the data that changed between two actions.)
- **Spark Datasource API** allows you to write your own code to ingest data from a custom source using the Spark datasource API and use a Hudi datasource to write as a Hudi dataset.

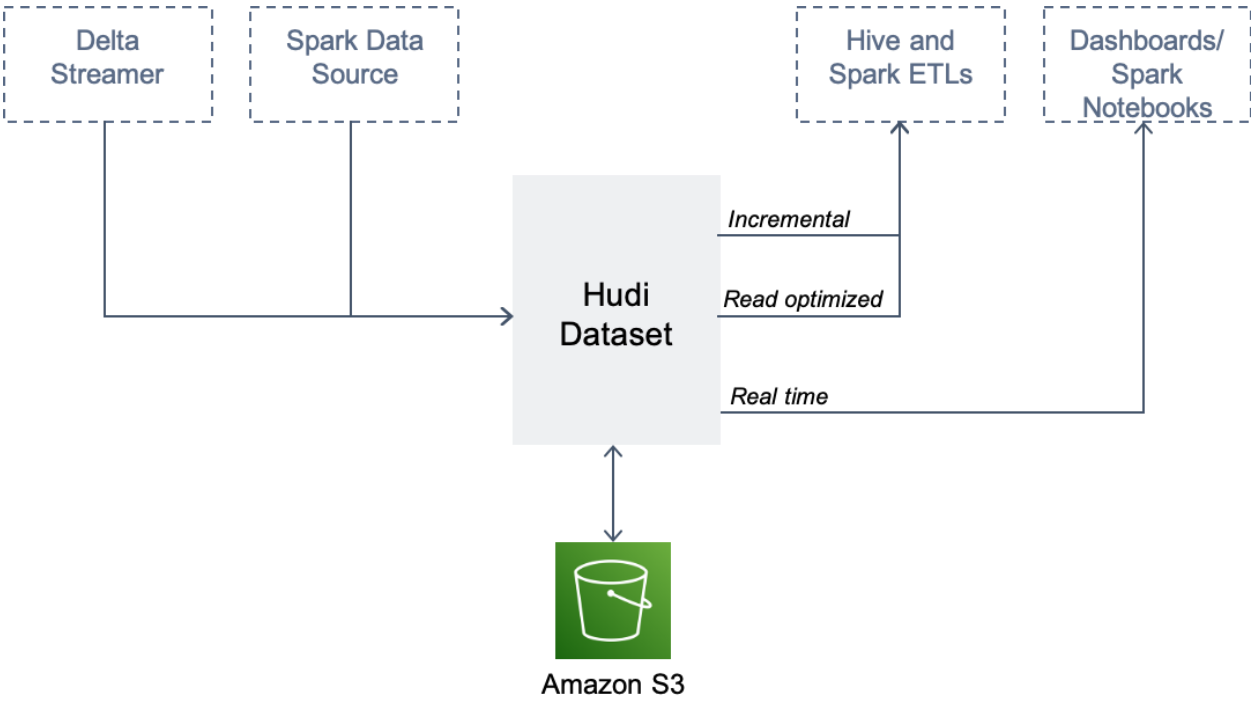


Figure 43: Writing Hudi datasets

The following table summarizes when to use the two write mechanisms with Hudi:

Table 6: Hudi write mechanisms

DeltaStreamer	Datasource API
Use Delta Streamer when you want a simple, self-managed ingestion tool that automates data compaction and provides automated checkpointing without the need to write any code	Use DataSource API when you are working with several varied data sources and want to create consolidated or derived tables – for example if you have existing Spark-based ETL pipelines you can use Hudi to take advantage of incremental pull, only reading the specific records that have changed.
Need to perform transformations on ingested data, for example dropping columns, casting or filtering data - Delta Streamer supports passing a SQL query template for SQL-based transformations, or allows you to plug in your own implementation for more advanced transformations.	You have existing data pipelines that need to work with both Hudi-managed and non-Hudi-managed datasets you can use the DataSource API.
Delta Streamer is also a good choice if you are ingesting data from Kafka, or using AWS DMS to land files in S3 and you don't want to have to write any code to apply those updates to your data set	Use Data Source API with Spark/Structured Streaming, allowing you to stream events into your Hudi dataset

Choosing the table type for workload

Hudi provides two storage options to choose from depending on whether your workload is read heavy (Copy on Write) or write heavy (Merge on Read). Use *Copy On Write* when:

- Your job is rewriting an entire table/partition to deal with updates
- Your workload is fairly steady and does not have sudden bursts
- You are already using Parquet files for your tables
- You want to keep things operationally simple, replace your existing Parquet files, and have no need for real-time views

Use *Merge on Read* when:

- You want ingested data available for query as soon as possible
- Your workload can have sudden spikes or changes in pattern
- You want collect the stream of changes and compact them periodically to help reduce write amplification (overhead)

Table 7: Comparison of Hudi table types

Table Type	Compactions	Write Throughput	Data Freshness SLA
Copy on Write	N/A		Low
Merge on Read	Inline	Low	High
Merge on Read	Offline		High

Querying Hudi datasets

During Hudi write operations with either DeltaStreamer or Datasource API, if Hive Sync is enabled, the dataset is available either as a Hive table or AWS Glue Data Catalog (depending on which one you specified when you launched your EMR cluster). The Hive table or AWS Glue Data Catalog that is created can be read using Hive QL, Spark SQL, or Presto. Hudi provides different logical views on the tables created. See Considerations and Limitations for Using Hudi on Amazon EMR for supported logical views for each of these query engines.

Table 8: Use cases for engines

Engine	Views	When to use?
Spark SQL	Read Optimized Real time Incremental	Notebooks & Data Science Machine Learning Custom data pipelines Best support for incremental & streaming ETL
Hive	Read Optimized Real time Incremental	Data warehousing ETL Incremental ETL pipelines
Presto	Read Optimized Real time	Interactive & ad hoc queries

Compaction

A compaction activity merges the log files with the base files to generate new compacted files written as a commit on Hudi's timeline. Compaction applies to the Merge on Read table type. There are two ways to perform the compaction – synchronous and asynchronous.

Synchronous (or inline) compaction is triggered at ingestion time after a commit or deltacommit as part of insert/upsert/bulk_insert operation. Use this compaction type when you want to quickly compact the recent 'N' partitions and can wait for delta logs to accumulate to merge into older partitions. As a result, your data lake will have the most recent data which is likely to be queried often.

Asynchronous compaction is run as a separate job that is either scheduled by an orchestrator or run manually. Use this compaction when you do not want to block the ingestion operation for compaction and require the compaction to be run as part of your workflow.

Deletes

Apache Hudi supports two types of record level deletes on data stored in Hudi datasets by specifying a different record payload implementation. Deletes can be performed using Hudi RDD API, Datasource API, and DeltaStreamer.

- **Soft Deletes:** Soft deletes let you keep the record key and null the values for all other fields. You can implement this by ensuring the appropriate fields are nullable in the dataset schema and simply upserting the dataset after setting these fields to null.
- **Hard Deletes:** Hard deletes are a stronger form of delete to physically remove any trace of the record from the dataset.

Performance and Tuning

Hudi is an Apache Spark based library. The general best practices for tuning Spark applications apply to Hudi workloads as well. Keep the following guidelines in mind when creating Hudi jobs on Amazon EMR.

- **EMR Cluster Size**: Use memory intensive nodes, such as Amazon EC2 R5 instances. Consider using Instance Fleets and Spot Instances to reduce costs. Aim to fit input data in memory if possible.
- **Input Parallelism**: Determines the number of files in your table. Make the property proportional to target files desired. Hudi defaults to (1500) which may be too high in certain instances.
- **File Size:** We recommend 128-512 MB file sizes by setting the `limitFileSize` (128 MB) property accordingly. The `compactionSmallFileSize` (100 MB) property defines size in MB that is considered as a "small file size". The default value should be appropriate for your application. The `parquetCompressionRatio (0.1)` property specifies expected compression of parquet data used by Hudi – adjust this property as needed. Note that with this property, you are balancing ingest/write latency vs read latency.
- **Off-heap memory:** Increase `spark.yarn.executor.memoryOverhead` or `spark.yarn.driver.memoryOverhead` if needed.
- **Spark Memory:** Reduce spark memory (`spark.memory.fraction`, `spark.memory.storageFraction`) conservatively if facing Out Of Memory errors allowing data to spill.
- **Bloom Filters:** Hudi assumes the `maxParquetFileSize` is 128 MB and `averageRecordSize` is 1024 B and hence approximately a total of 130K records in a file. Set the property `bloomFilterNumEntries (60000)` to half the number of records in a file. The trade off is disk space for lower false positives.

Table 9: Properties

Properties	Description
hoodie. [insert|upsert|bulkinsert].shuffle.parallelism	Parallelism determines the initial number of files in your table.
hoodie.parquet.max.file.size	Target size for parquet files produced by Hudi write phases.
hoodie.parquet.small.file.limit	This should be less than maxFileSize.
hoodie.parquet.compression.ratio	Expected compression of parquet data used by Hudi.

hoodie.consistency.check.enabled	Additional check to handle S3's eventual consistency model.

Sample Architecture

The following architecture shows the workflow for ingesting upserts and deletes using Apache Hudi on Amazon EMR.

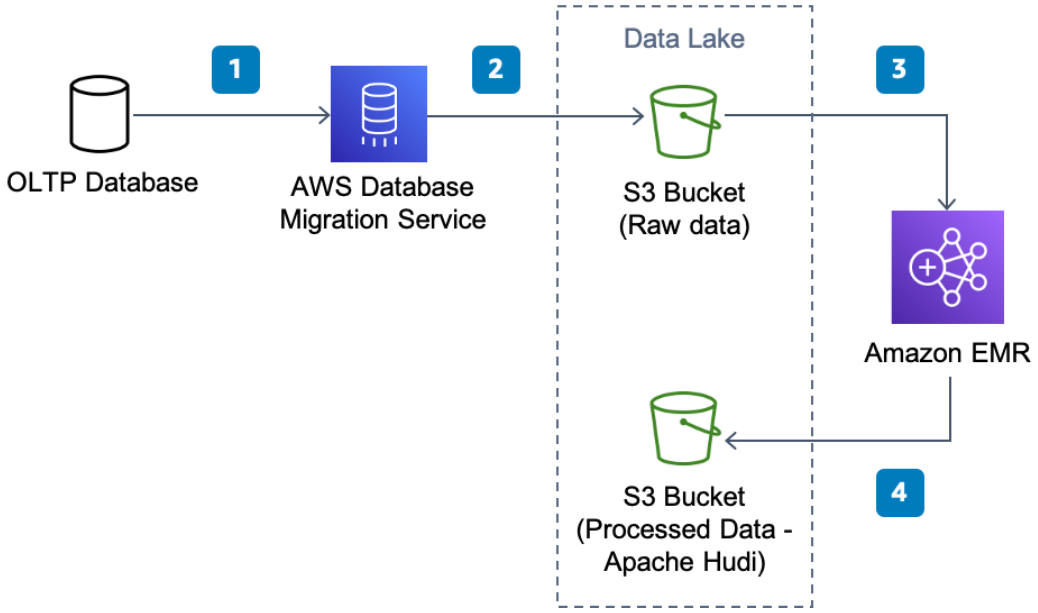


Figure 44: Ingesting upserts/deletes with Apache Hudi on Amazon EMR

1. Ingest full load and CDCs from OLTP databases using AWS Database Migration Service.
2. AWS DMS task deposits full load and CDCs to Amazon S3.
3. Ingest changes dropped into Amazon S3 incrementally.
4. Write Apache data formats to Amazon S3.

The Amazon EMR workload differs depending on your Hudi table type:

- *Copy on Write:* Amazon EMR uses a batch workload where transient EMR clusters (on Instance Fleet and Spot instances) run periodically (e.g. every hour or end of day).
- *Merge on Read:* Amazon EMR uses a streaming workload where persistent EMR clusters run Spark Streaming or Apache Hudi DeltaStreamer jobs continuously.

Providing Ad Hoc Query Capabilities

Considerations for Presto

Choosing between Amazon Athena and PrestoDB

Amazon Athena is a serverless interactive query engine that executes SQL queries on data that rests in Amazon S3. Many customers use Athena for a wide variety of use cases, including interactive querying of data to exploring data, to powering dashboards on top of operational metrics saved on S3, to powering visualization tools, such as Amazon QuickSight or Tableau. We strongly recommend that you consider Amazon Athena for these types of workloads. Athena is easy to integrate with, has several features, such as cost management and security controls, and requires little capacity planning. All of these characteristics lead to lower operational burden and costs. However, there are some use cases where PrestoDB may be better suited than Amazon Athena. For example, consider the following priorities:

- **Cost reduction:** If cost reduction is your primary goal, we recommend that you estimate cost based on both approaches. You may find that the load and query patterns are cheaper to run using Presto on Amazon EMR. See if cost increases, if any, outweigh the benefits of running and maintaining a Presto cluster on EMR that is able to scale and provides availability, versus the features that Amazon Athena provides.
- **Performance requirements:** If your use case includes a high sensitivity to performance choose to fine-tune a Presto cluster to meet the performance requirements.
- **Critical features:** If there are features that Amazon Athena does not currently provide, such as the use of custom serializer/deserializers for custom data types, or connectors to data stores other than those currently supported, then Presto on EMR may be a better fit.

For performance tips and best practices for Athena and Presto, see Top 10 Performance Tuning Tips for Amazon Athena on the *AWS Big Data Blog*.

Metadata Management

AWS Glue as a Data Catalog

Starting with Amazon EMR release 5.10.0, Amazon EMR can use AWS Glue Data Catalog as the default Hive metastore for Presto. See the Data Catalog Migration section for more information on this approach. The benefits included in that section also apply to Presto running on Amazon EMR.

When using AWS Glue Data Catalog with Presto on Amazon EMR, the authorization mechanism (such as Hive SQL authorization) is replaced with AWS Glue Based Policies.

You are also required to separately secure the underlying data in Amazon S3. You can secure this data by using an S3 bucket policy or AWS IAM policy. You may find it more efficient to use IAM policies as you can centralize access to both Amazon S3 and the AWS Glue Data Catalog.

For more information on using the AWS Glue Data Catalog, see Easily manage table metadata for Presto running on Amazon EMR using the AWS Glue Data Catalog on the AWS Big Data Blog. For existing limitations on the interaction between Presto and AWS Glue Data Catalog, see Considerations When Using AWS Glue Data Catalog.

EMRFS and PrestoS3FileSystem Configuration

By default, Presto running on Amazon EMR release version 5.12.0 or later can use EMRFS to access data on Amazon S3. Presto running on previous releases of EMR only uses the PrestoS3FileSystem, a component of the Hive connector.

Accessing data via EMRFS allows you to configure Amazon S3 encryption requirements in an EMR Security Configuration. With EMRFS, you can also use separate IAM roles within your Presto Cluster to control access to data according to a user, group, or Amazon S3 location.

For EMR release version 5.12.0 or later, you can switch from using EMRFS to using the PrestoS3FileSystem. This approach may be beneficial if your organization is still relying on Hive SQL Based authorization and Hive metastore running on an RDMBS. For additional details on configuring the PrestoS3FileSystem on Amazon EMR, see EMRFS and PrestoS3FileSystem Configuration.

HBase Workloads on Amazon EMR

Apache HBase on Amazon S3 using Amazon EMR can introduce significant cost savings for read-heavy workloads.[6] Due to the ability to use Amazon S3 as the primary storage mechanism, the HBase cluster can be much smaller than typical HBase deployments.

For details on tuning HBase for best performance on Amazon S3, see Migrate to Apache HBase on Amazon S3 on Amazon EMR: Guidelines and Best Practices on the AWS Big Data Blog.

HBase Upgrades

For HBase upgrades, we recommend that you run the newer version of HBase alongside the previous version until all testing is complete. Take a snapshot from HBase and use that snapshot to start a new cluster with the upgraded version. You can run both clusters simultaneously and perform updates and reads from both clusters. When testing is complete, you can shut down the old cluster. If you encounter any issues that require rolling back the upgrade, you can move back to the old cluster.

Optimize Bulk Loads on HBase on S3

A common approach when migrating an existing cluster to HBase on S3 using EMR is to use bulk loads. Although this approach can be an effective way to bootstrap your new cluster, there are several ways you can optimize bulk loads for the best performance.

Depending on your version of HBase and where your generated StoreFiles are, the command to perform the bulk load is similar to the following code:

```
hbase org.apache.hadoop.hbase.mapreduce.LoadIncrementalHFiles \
  <s3://bucket/storefileoutput/> <tablename>
```

The command initiates the following process:

1. Lists available StoreFiles in the storefileoutput location
2. Determines in which Region a StoreFile should be placed and if that StoreFile fits in the given Region.
3. If the StoreFile is too large for the Region (based on `hbase.hregion.max.filesize`), or if the Region has split since the files were created, the master node splits the StoreFile into two files and add these two new files back into the list of StoreFiles to process.
4. For each Region that has StoreFiles to load, a request is issued to the necessary RegionServer to initiate a BulkLoad.
5. The RegionServer copies the StoreFile to the target file system, then loads the StoreFile into HBase.

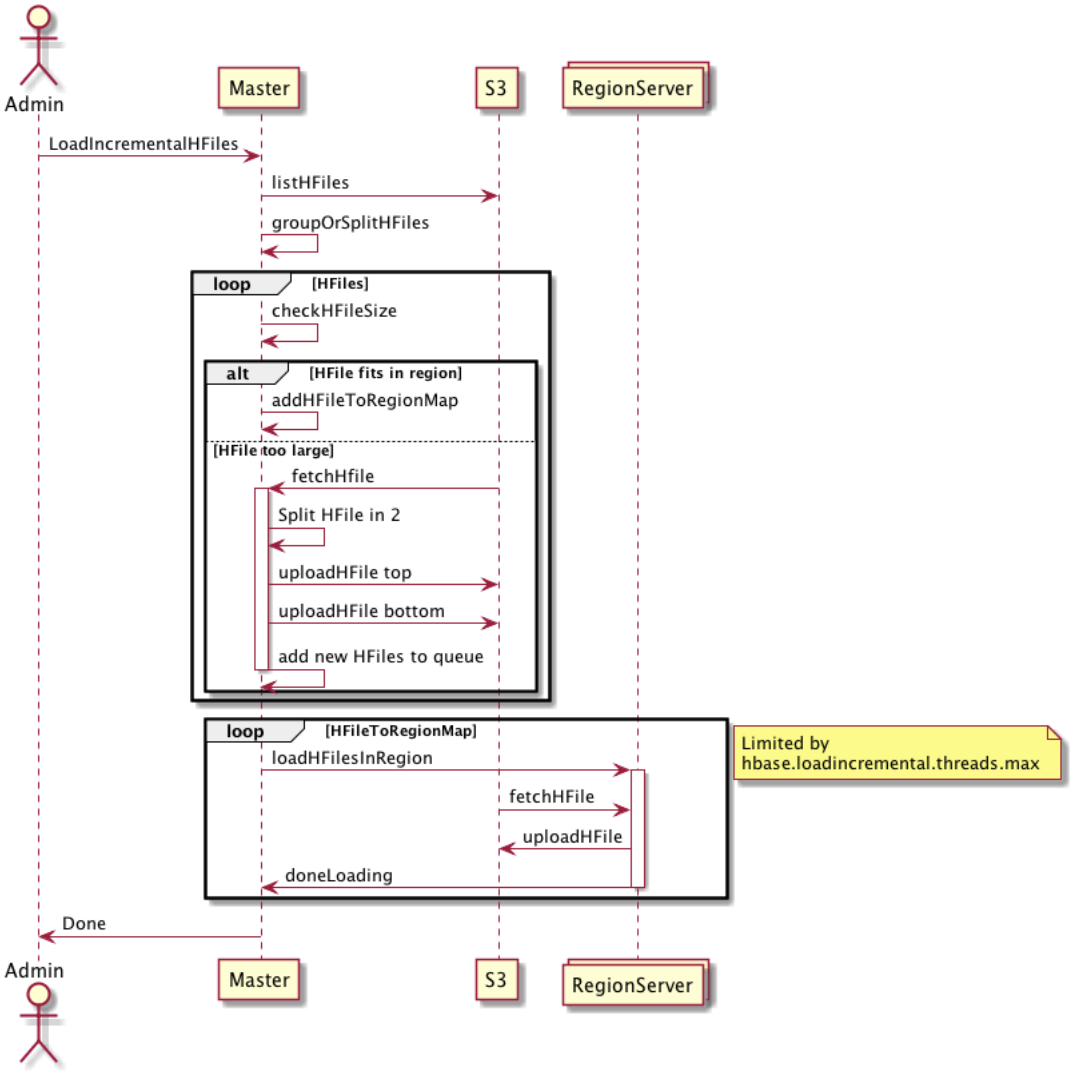


Figure 45: Overview of CompleteBulkLoad process

As with any distributed system, the performance of this process depends on available CPUs and network bandwidth.

Optimize S3 Uploads

You can optimize S3 uploads by adjusting size settings:

- **fs.s3.threadpool.size**: By default, this setting is 20, but you can increase it to increase the number of parallel multipart uploads that are occurring. You may need to use an instance with a higher number of CPUs to increase this setting.
- **fs.s3n.multipart.uploads.split.size**: Increase setting when you're uploading large files to help avoid reaching multipart limits.

Increase Threads on the Primary Node

The primary node (or whichever node you run LoadIncrementalHFiles on) performs two primary steps: splitting StoreFiles (if they don't fit in the Region) and coordinating the BulkLoads with each responsible RegionServer. These actions happen in a thread pool that is by default the size of the number of available CPUs on the node. For example, if your master node is a small m4.large with only two vCPUs, your entire bulk load process can be blocked if two StoreFiles must be split. You can address this issue in two ways:

- Use a larger node with more vCPUs.
- Define `hbase.loadincremental.threads.max` variable when running the job

However, even if you have 100 RegionServers, the LoadIncrementalHFiles command is only be able to use as many of them as is defined in the `hbase.loadincremental.threads.max` variable. If your node has eight vCPUs, only eight region servers are used. You can likely increase that variable to the number of RegionServers you have, but if your master must split StoreFiles, it could get bogged down quickly.

Increase RPC Timeout

If you have generated large StoreFiles (upwards of 10 GB) or if you have a high number of StoreFiles for a specific RegionServer, your LoadIncrementalHFiles command may occasionally return an "Error connecting to server" message due to a CallTimeoutException. This behavior is normal, but the issue results in multiple attempts to perform the BulkLoad and typically results in a failed load.

To work around this issue, you can increase the RPC timeout by using the `hbase.rpc.timeout` variable when starting your job.

Monitoring your Bulk Load

While the bulk load is running, the HBase UI does not include enough metrics to carefully monitor the bulk load. For better insight into the bulk load, start your EMR cluster with Ganglia. In addition to general CPU and network metrics, Ganglia includes a number of HBase metrics. Specifically, `regionserver.Server.Bulkload_count` can show you how many bulk loads have finished across each RegionServer.

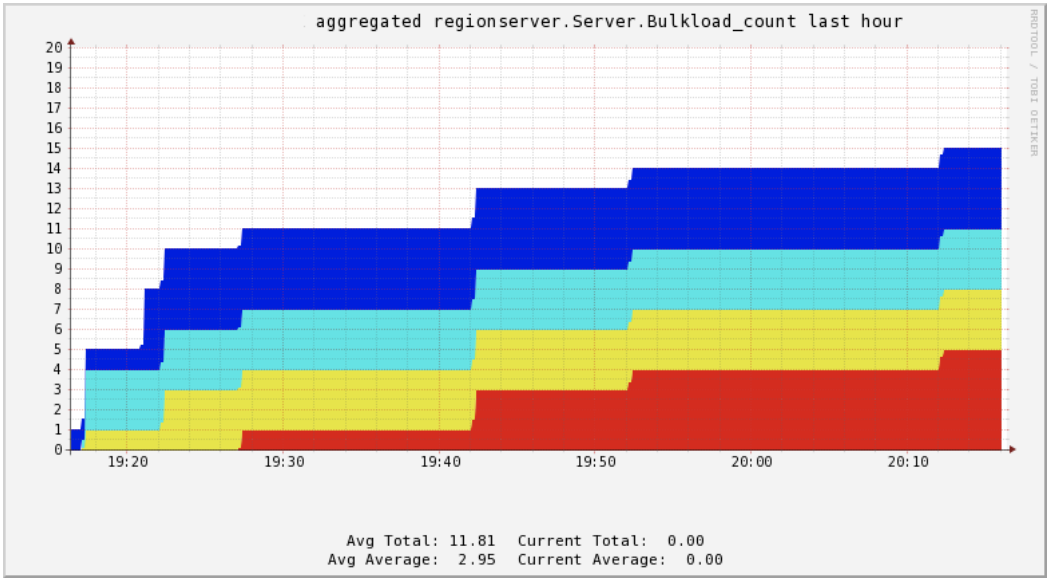


Figure 46: Ganglia region server bulk load count

Debugging your Bulk Load

If you run into issues, you can enable debugging on HBase to for more detailed information on those issues. To enable debugging, set the `log4j.logger.org.apache.hadoop.hbase.mapreduce.LoadIncrementalHFiles` variable to `DEBUG`.

Summary and Examples of Optimization Strategies

This section included three ways to optimize bulk loads:

- Make the best available use of Amazon S3 and network bandwidth.
- Make the best available use of CPU on the master node.
- Change HBase settings for our expected workload.

For all of the configuration variables above, you can specify these as command-line variables to LoadIncrementalHFiles, or you can configure these settings cluster-wide using various Hadoop configuration files. See the following examples.

Increasing Threads and RPC Timeout

This command shows setting the number of threads to 20 and the HBase RPC timeout to 10 minutes.

```
hbase org.apache.hadoop.hbase.mapreduce.LoadIncrementalHFiles \
  -Dhbase.loadincremental.threads.max=20 \
  -Dhbase.rpc.timeout=600000 \
```

```
  <s3://bucket/storefileoutput/> \
  <tablename>
```

Define HBase on S3 Settings and Enable Debugging

If you create your cluster through the AWS CLI, the JSON file below shows a typical configuration file. The file defines HBase on S3 settings and enables DEBUG and TRACE logging on two HBase classes.

```
[
    {
        "Classification": "hbase",
        "Properties": {
            "hbase.emr.storageMode": "s3"
        }
    },
    {
        "Classification": "hbase-site",
        "Properties": {
            "hbase.rootdir": "s3://<bucket>/<hbaseroot>",
            "hbase.hregion.max.filesize": "21474836480",
        }
    },
    {
        "Classification": "hbase-log4j",
        "Properties": {

"log4j.logger.org.apache.hadoop.hbase.mapreduce.LoadIncrementalHFil
es": "DEBUG",
            "log4j.logger.org.apache.hadoop.hbase.ipc.RpcServer":
"TRACE"
        }
    }
]
```

Migrating Apache Impala

If you are running Apache Impala and are looking to migrate it to Amazon EMR, we recommend that you use Amazon Athena or Presto. See Migrating Presto for more details on how to choose Amazon Athena over PrestoDB.

If you must use Impala due to a use case that is not covered by Presto or Athena, then you have three options to install Impala:

- Manually install Impala on Amazon EC2.
- Use a bootstrap action to install Impala on Amazon EMR.

- Use a third-party cloud provider that installs and manages Impala.

Because Impala is not a managed application, AWS Support and Amazon EMR service teams are not able to support an Impala installation.

Operational Excellence

Upgrading Amazon EMR Versions

One best practice is to upgrade your Amazon EMR releases in a regular cadence. Upgrading your clusters' software ensures that you are using the latest and greatest features from open source applications. The following are a few benefits of staying up-to-date with software upgrades:

- Performance enhancements enable applications to run faster.
- Bug fixes make the infrastructure more stable.
- Security patches help keep your cluster secure.

These benefits apply to both the open source application software and the Amazon EMR software needed to manage the open source software.

The following figure is a sample of Amazon EMR 5.x software releases and corresponding open source application versions from July 2019 through January 2020. At the time of this document, Amazon EMR releases a new version approximately every 4–6 weeks, which pulls the latest version of the software. For complete list of releases and release notes, see Amazon EMR 5.x Release Versions.

Figure 47: Sample of Amazon EMR 5.x Release Versions

See Software Patching for recommendations on when it may be appropriate to patch software on your Amazon EMR cluster.

Upgrade Process

When upgrading software, the risk of refactoring exists in terms of performance and data quality. Upgrades may change API interfaces so that your code may no longer run as is on the new framework. Upgrades can also introduce new bugs, which can cause applications to fail. AWS provides a best effort to identify regressions in open source software before Amazon EMR releases by running a large suite of integrations tests but some regressions may be difficult to identity. Therefore, it is imperative that each release is tested before making it available to your users. However, the more often you upgrade, the smaller number of changes between versions, which reduces the effort in upgrading as the risk of regressions is reduced.

Recommended Upgrade Steps

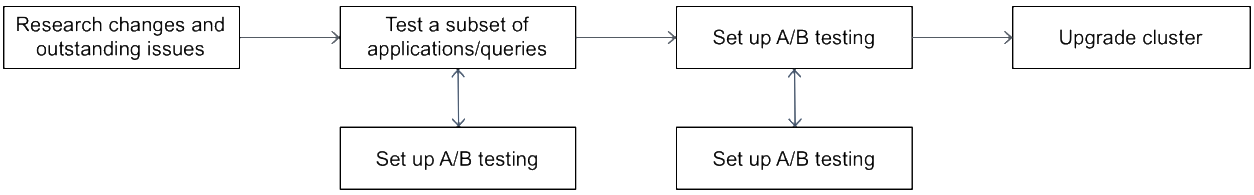


Figure 48: Recommended upgrade steps

Research Changes and Outstanding Issues

All open source applications have release notes available and most provide JIRA for issue tracking. Before an upgrade, you can save time by doing research to look for bugs, issues, or configuration updates.

The following table lists common open source applications, their release notes, and issue tracking systems. For Amazon EMR, see Amazon EMR 5.x Release Versions.

Table 10: Application Links for Release Notes and Issue Tracking

Application	Release Notes	Issue Tracking
Apache Hadoop	https://hadoop.apache.org/old/releases.html	https://issues.apache.org/jira/projects/HADOOP/issues
Apache Hive	https://hive.apache.org/downloads.html	https://issues.apache.org/jira/projects/HIVE/issues
Apache Spark	https://spark.apache.org/releases/	https://issues.apache.org/jira/projects/SPARK/issues
Presto	https://prestodb.github.io/docs/current/release.html	https://github.com/prestodb/presto/issues

Application	Release Notes	Issue Tracking
Apache HBase	https://hbase.apache.org/downloads.html	https://issues.apache.org/jira/projects/HBASE/issues

Test a Subset of Applications/Queries

Before users test the new release or configuration, we recommend that you test the version with a subset of use cases that is representative of the overall usage. This approach ensures that any configuration issues are caught before deployment.

Fix Issues

If you find an issue when testing a version, follow these steps:

1. Check if a configuration value can fix the issue. For example, see if you can use a configuration value to disable a problematic new feature or enhancement.
2. Check if the issue has already been identified and fixed in a later version of the open source project. If there is a fix in a later version, notify an AWS Support engineer through AWS Support channel. AWS will evaluate if it can be included in our next release.
3. Change the application or query to avoid the issue.
4. Contact AWS Support to see if any workarounds exist.
5. Abandon the upgrade if there is no workaround and wait for a release that has the required fix.

Set up A/B Testing (Recommended)

The next step is to gradually move the workload to the new configuration. This approach provides you with the option to abort the upgrade if a serious issue is found in your production environment. If you are using Amazon EMR for interactive user querying, setting up a router helps move the load from one cluster to another in a controlled fashion (*Figure 49*). You can also use a load balancer that supports both traffic weighting and sticky sessions.

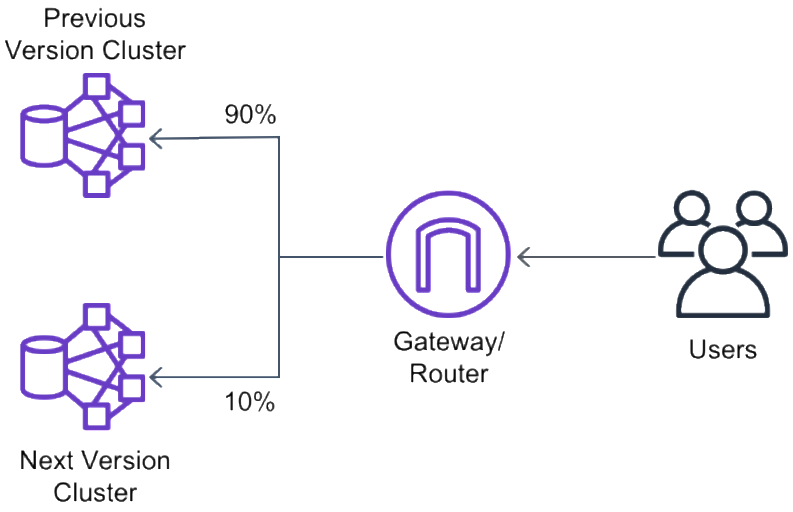


Figure 49: Using a router to gradually move load

Complete Upgrade

Complete your upgrade by moving all of your Amazon EMR clusters to the new version. Finally, discontinue use of the older version.

Best Practices for Upgrading

- Upgrades require time and effort – make sure that your teams schedule upgrades and allow for the time it takes to complete upgrades.
- Be aware of dependencies that can change when upgrading.
- When performing manual testing, replicate your Hive metastore to ensure that the schema remains backward compatible.
- If you can, track performance of the jobs to ensure that a significant regression has not occurred.
- Split your clusters by applications. This approach allows you to upgrade components individually, rather than as a package.
- Research what has changed between releases so that issues are easier to identify.
- Use Amazon Route 53 to automatically register clusters. This approach makes it easier for users to point to them. For more information on setting up Amazon Route 53, see Dynamically Create Friendly URLs for Your Amazon EMR Web Interfaces on the AWS Big Data Blog.

General Best Practices for Operational Excellence

Configure all of your EMR clusters as transient clusters

Configuring all of your Amazon EMR clusters as transient clusters has many benefits. Transient clusters are clusters that are started, perform some work and then shut down. For clusters to be transient, state cannot be kept on the cluster as the state is removed when the cluster is shut down. This approach makes disaster recovery easier and quicker, makes Amazon EMR upgrades easier and quicker, and allows you to separate your storage and compute resources for independent scaling.

Provision your resources in an automated way

Use AWS CloudFormation scripts that call AWS Command Line Interface (AWS CLI) or SDK code to automatically provision EMR clusters. This approach allows the clusters to be created consistently and for cluster changes to be tracked more easily.

Ensure that all of your code, scripts, and other artifacts exist within source control

By ensuring all of your code, scripts, and other artifacts exist within source control, you can track what has changed over time. In addition, you can track down the changes that may have impacted your environment's operation. This approach also allows you to deploy the artifacts in different stages, such as a beta environment, and reduces the chances that a bad artifact moves from one stage to another.

Monitor your clusters for abnormal behavior and employ automatic scaling

Specifically, make sure to watch for long running or stuck jobs as well as disk or HDFS capacity issues.

Consider using AWS Glue, Amazon Redshift, or Amazon Athena

Although Amazon EMR is flexible and provides the greatest amount of customization and control, there is an associated cost of managing Amazon EMR clusters, upgrades, and so on. Consider using other managed AWS services that fulfill your requirements as they may have lower operational burden, and in some cases, lower costs. If one of these services does not meet your use case requirements, then use Amazon EMR.

Testing and Validation

After you have established a process for migrating your data from your source systems and have analytics jobs running, ensuring that your jobs are getting good data and defining rules around your data becomes increasingly important.

Unit tests are usually written for code, but testing data quality is often overlooked. Incorrect or malformed data can have an impact on production systems. Data quality issues include the following scenarios:

- Missing values can lead to failures in the production system that require non-null values.
- Changes in the distribution of data can lead to unexpected outputs of machine learning models.
- Aggregations of incorrect data can lead to ill-informed business decisions.

This chapter covers multiple methods of checking data quality, but the approaches covered here are not exhaustive. There are many third-party vendors and partners that offer solutions around data profiling and data quality, but these are out of scope for this document. The approaches and best practices described here can and should be applied to address data quality.

Data Quality Overview

Data quality is essentially the measurement of how well your data meets the expectations of its consumers. Stated another way, it is the correctness of the data relative to the construct or object that it is trying to model. Because this is a hard concept to define for data in general, consider data quality from the perspective of the following dimensions:

- **Completeness** measures how comprehensive your data is.
- **Uniqueness** looks at duplication within your data.
- **Timeliness** ensures that your data is fresh and available within your requirements/SLAs.
- **Validity** measures how well your data conforms to defined schemas, syntax, and requirements.
- **Accuracy** measures how correct your data is in representing objects within the scope of requirements.
- **Consistency** evaluates if your data has differences when looking at references to the same object or relationships between objects.

This is not a complete list of data dimensions. Organizations can use other metrics and attributes as dimensions for their own data quality needs.

Check your Ingestion Pipeline

Data integrity can be considered a crucial part of data quality checks. One of the most important areas to check is whether the ingested raw data is correct.

To validate that data was migrated correctly from the source system to the target system, you should first understand the existing mechanism that your tool uses for ensuring that data is valid. For example, consider Apache Sqoop. Sqoop validates a data copy job through the use of the `--validate` flag, which performs a row count comparison between the source and target. For example:

```
$ sqoop import --connect jdbc:mysql://db.foo.com/corp --table
WIDGETS --validate
```

However, there are various limitations with how Sqoop performs this validation. Sqoop does not support:

- An all-tables option
- A free-form query option
- Data imported into Hive or HBase
- A table import with `--where` argument
- Incremental imports

Although some of these issues can be mitigated, the underlying problem is that Sqoop fundamentally only performs row count comparisons. To validate every single value transferred, you must reverse your data flow with Sqoop and push the freshly copied data back to the source system. After this reverse push, you can compare the old data against the new data with hashes or checksums. Because this is an expensive operation, this scenario is one where you must consider risk acceptance and data policy.

As part of your migration, you must determine the level of certainty around data accuracy and have a corresponding and quantifiable metric for it. The stringency of your requirements determines cost, complexity, and performance implications of your migration strategy, and could possibly impact the performance of your overall solution. In addition, depending on the level of acceptable risk, alternate approaches, such as sampling or application-level validation, may be viable options.

Another example is with the AWS CLI, where questions around data quality (specifically, integrity) often arise when using it to transfer data from source systems to Amazon S3. It is important to understand the characteristics of your target destination as well as exactly how the AWS CLI and other tools help validate data that is copied. This way, you can establish reasonable data quality goals and thoroughly address and answer questions from your architecture teams and business owners.

The following list is some common questions that arise when you use AWS CLI and other data movement tools. Specifically:

- Does the AWS CLI validate objects as they land into Amazon S3?
- How does the AWS CLI validate the objects?
- Does Amazon S3 expose the checksum used?
- What happens in the case of multi-part uploads in Amazon S3? How do you calculate a checksum for parts as well as the whole object?
- How can you validate that an uploaded object's checksum is accurate without relying on the Amazon S3 ETag?

Overall Data Quality Policy

Frequently, the sources of data, and the processes used to extract them while building and testing initial analytics jobs and models, are different from those used in production for analytics or inferencing. For example, initial research (asking "is it possible?" and data discovery) is frequently performed on sample, cleaned, simplistic, and possibly enriched data extracted from a data lake or established data sources, for convenience and speed of access. The implicit assumption is that the data operated on in production is the same and can be provided quickly enough to act on, but this scenario doesn't always apply; especially in environments where there is a firm boundary between development and production environments. This assumption should be tested to ensure that the analytics job or machine learning model will work as expected. Create a diagram of the data pipeline used to build the model showing where all data is from and how it is transformed. This diagram can help you identify potential challenges.

Also, create diagrams of the data pipeline and data catalog to be used in production. Make note of the similarities and differences between the two pipelines.

- If they are the same, then they are likely subject to the same errors. Are these errors important?
- If they are not the same, then different sources or different processing has been applied. Do any of these differences impact your analytics jobs or ML models? How do you know?

Share the diagrams and summaries with subject matter experts and project sponsors. Discuss the differences and get agreement that they appear reasonable to all stakeholders. If the gap between the two pipelines is found to be too large (a subjective assessment), consider other approaches to sourcing data that are more representative of the data in production circumstances.

The more data sources that are involved, the more disparate the data sources that are to be merged. Similarly, the more data transformation steps that are involved, the more complex the data quality challenge becomes.

Estimating Impact of Data Quality

During initial analytics jobs, it is usual to begin with cleaned data (for example, from a data lake), or to clean data before running the initial jobs. For example:

- When merging data, data might be dropped if no direct key match is found.
- Records with null or extreme values might be dropped.

Frequently, there are many individual cleaning and transformation steps performed before the data is used for analytics or ML training. However, when the job or model is used in production, the data used is generally coming from a different source (i.e. production data), and comes to the model's inference endpoint through a different production-focused path. Your analytics job, ETL workflow, or ML model was built to work well for cleaned data inputs off of the development or test dataset. To ensure that your job in production behaves the same as it did in a lower environment, consider comparing statistics and validating the model against unclean data inputs.

Compare Statistics

To ensure that the analytics job or model performs well in production, add a formal checkpoint that compares the source input data to the data job model actually used to train on. Make sure to evaluate the data from both a quantitative and qualitative perspective.

Quantitative Evaluation

In your quantitative evaluation, review counts, data durations, and the precision of the data inputs in addition to any other quantifiable indicators that you have defined.

Compare counts to identify, track, and highlight data loss, and test against what seems reasonable. What percentage of source data was used to actually build and test the model? Is there any potential bias as a result of unintentionally dropped data from a merge? Is the data storage subsystem filtering, averaging, or aggregating results after some time period (for example, for log messages)?

Review data duration and retention to determine what time period each dataset covers. Are all of the potentially relevant business cycles included, especially during peak load?

Quantify precision, by comparing the mean, median, and standard deviation of the data source and the data used to train the model. Calculate the number or percentage of outliers. For lower dimensional data or key variables, box plots can provide a quick visual assessment of reasonableness. Quantifying precision in this way helps you to scope out what 'expected' values should be and build out alerts to notify you for data that is outside of this scope.

Qualitative Evaluation

Accuracy is equally important as precision, but likely can only be assessed qualitatively. For example, based on experience and sample exploration, how confident are you that the data is accurate? Are there sufficient anecdotes of errors? For example, do operators report this sensor is always running high? The

actions to take based on this evaluation vary widely. A frequent result is to segment the data based on some factor discovered during the analysis and take a different action on each segment. For example, in a study of intelligent transport systems, a set of sensors were identified as misconfigured and routed to be repaired, whereas another subset was used in traffic analysis.[7]

Validate Model Against Unclean Data Inputs

A simple but powerful technique to validate your data model is to take a subset of data that was eliminated during every cleaning or transformation step from the raw data and compare it to the data eventually used to run the job. Then, assess the resulting outputs. Does the endpoint provide reasonable responses in all cases? Use the results to identify where checks and error handling should be added. Should error handling be added to the inference endpoint? Or, should the applications that are calling the jobs be required to identify and remove problematic inputs, or handle problematic outputs?

Tools to Help with Data Quality

Instead of having to implement all of this on your own, here are some vendor and open source tools to help you with the data quality process:

Apache Griffin

Apache Griffin is an open source data quality solution for big data that supports both batch and streaming modes. It offers a unified process to measure your data quality from different perspectives, helping you build trusted data assets and therefore boosting your confidence for your business.

You can combine Griffin with tools such as StreamSets or Apache Kafka to have an end-to-end streaming workflow that performs data quality checks for you.

Deequ

Deequ is a library built on top of Apache Spark for defining "unit tests for data", which measure data quality in large datasets. It allows you to calculate data quality metrics on your dataset, define and verify data quality constraints, and be informed about changes in the data distribution. Instead of implementing checks and verification algorithms on your own, you can focus on describing how your data should look. Deequ is implemented on top of Apache Spark and is designed to scale with large datasets that typically live in a distributed filesystem or a data warehouse.

Amazon EMR on AWS Outposts

AWS Outposts is a fully managed service that extends AWS infrastructure, AWS services, APIs and tools to virtually any datacenter or on-premises facility for a truly consistent hybrid experience. AWS Outposts offers you the same AWS hardware infrastructure to build and run your applications on-premises and in the cloud. You can also run Amazon EC2, Amazon EBS, Amazon EKS, Amazon RDS and analytics services such as Amazon EMR on-premises.

Outposts are connected to the nearest AWS Region to provide the same management and control plane services on-premises, and you can access the full range of AWS services available in the Region to build, manage, and scale your on-premises applications. For more information about AWS Outposts, see the AWS Outposts User Guide.

Amazon EMR on AWS Outposts is ideal for low latency workloads that need to be run in close proximity to on-premises data and applications. Creating an Amazon EMR cluster on AWS Outposts is similar to creating an Amazon EMR cluster in an AWS Region. Starting with Amazon EMR version 5.28.0, you can create and run EMR clusters on AWS Outposts. You can seamlessly extend your VPC on-premises by creating a subnet and associating it with an Outpost just as you associate subnets with an Availability Zone in the cloud.

Limitations and Considerations

Network connectivity between an AWS Region and an Outpost plays an important part in an EMR cluster. In the event of an absence of connectivity, consider the following aspects of the EMR service:

- Existing clusters continue to run, however no new clusters can be created
- New actions on existing clusters will fail, such as replacing an unavailable instance
- Other actions will be delayed, such as adding new steps to a running cluster or checking step status or sending CloudWatch metrics
- Existing clusters with terminate protection disabled may be terminated. Make sure to Enable terminate protection.
- Other external dependencies such as Amazon S3, DynamoDB, and Amazon RDS will not be accessible
- Spot Instances are not available for Amazon EMR on AWS Outposts. Only on-demand EC2 instances are supported.
- Only General Purpose SSD (GP2) Amazon EBS volumes are supported for additional storage.
- Not all instance types are supported by Amazon EMR on AWS Outposts.

For supported instance types and more information, see EMR Clusters on AWS Outposts in the *Amazon EMR Management Guide*.

Support for Your Migration

Amazon EMR Migration Program

The Amazon EMR migration program has two main deliverables. First, this migration guide, and second, a free workshop to help you plan your migration.

Migration to EMR Workshop

The Migration to EMR Workshop is two-day, customizable onsite workshop that can jump-start your migration to the AWS Cloud. The workshop provides a small and interactive setup where participants can interact directly with AWS experts, discuss strategies, and plot a way forward. The workshop focuses on explaining the benefits of a cloud architecture, dives deep into benefits of Amazon EMR, common usage patterns, architecting a data lake on AWS, migration methodologies, and security controls. The workshop also has guided hands-on labs that allow participants to try the most common architecture patterns in the cloud.

Amazon EMR is one of the largest Spark and Hadoop service providers in the world, enabling customers to run ETL, machine learning, real-time processing, data science, and low-latency SQL at petabyte scale. At the end of the session participants will have an understanding of

- Successful migration strategies followed by large customers who have migrated from on-premises clusters.
- Design patterns, such as using Amazon S3 instead of HDFS, taking advantage of both long- and short-lived clusters, using completely managed notebooks, and other best practices.
- Lowering cost and cloud-scale elasticity with AWS Auto Scaling and Spot Instances
- Security best practices with end-to-end encryption, authorization and fine-grained access control.

For more information or to schedule your EMR migration workshop, visit Apache Hadoop and Apache Spark to Amazon EMR Migration Acceleration Program.

AWS Professional Services

AWS Professional Services provides a global specialty practice to support focused areas of enterprise cloud computing. Specialty practices deliver targeted guidance through best practices, frameworks, tools, and services across solution, technology, and industry subject areas. Their deep expertise helps you take advantage of business and technical benefits available in the AWS Cloud.

Specific to Amazon EMR, the domain expertise of the Data Analytics practice helps organizations derive more value from their data assets using AWS services. Specific to Hadoop Migrations, AWS Professional

Services has a prescriptive and proven methodology. This methodology includes an alignment phase and launch phase, which are both described in the following sections.

Hadoop Migration Alignment to Amazon EMR and Amazon S3 Data Lake (and AWS Stack)

AWS Professional Services partners with you to learn and document your current state environment and the desired future state outcomes. Although the scope of this phase is mutually confirmed, the following list is activities suggested to be covered during the Hadoop migration alignment phase.

- Business understanding: Confirm business drivers and success metrics.
- Identification and documentation of value drivers to inform future state architecture decisions. Considerations given to internal SLA targets and existing and new business value.
- Data prioritization: Use business drivers and technical considerations as prioritization criteria for the defined scope.
- Data assessment: Assess end-to-end data flow architecture to recommend a future state Hadoop environment inclusive of source and target destinations, file formats, and so on.
- Technical understanding: Deconstruct Hadoop workloads to determine a migration path to Amazon EMR.
- Hands-on labs for existing application mapping (e.g., Impala to Presto) or use cases (e.g., data science notebooks)
- Security deep dive (Ranger, Sentry, Kerberos, LDAP, IAM)
- Confirm AWS Glue or Amazon RDS Data Catalog options.
- EMR service pricing exercise/what-if analysis
- Provide a detailed work plan for delivery of a migration to Amazon EMR.

Given the dependency Amazon EMR has with Amazon S3, the second section of the alignment phase focuses on the S3 data lake.

- AWS Lake Formation service introduction for future-proofing the architecture inclusive of serverless capabilities, as interested. Services include Amazon QuickSight and AWS Lake Formation plus serverless services AWS Glue and Amazon Redshift Athena.
- Provide recommendations on data ingestion architecture.
- Provide recommendation on data storage architecture approach that meets security, control and access requirements.
- Provide recommendations on Data Catalog approach.
- Provide recommendation on data serving layer for downstream applications or users for access to the catalog and perform data exploration for self-service.

- Provide recommendations for on boarding users to the platform to enable ease and reusability.
- Provide recommendations on decoupling compute and storage for a cost optimized data lake.

Hadoop Migration Alignment and S3 data lake outcomes:

- Identification of up to five use cases
- Closure Architecture document aligning requirements to architectural design
- Detailed migration path for execution of the EMR Launch work
- AWS Services Cost Estimate *(Platform Spend)*
- Closure Architecture document aligning requirements to architectural design
- Recommendations on picking right tools for job based on usage
- S3 bucket strategy built on AWS for workloads defined

Amazon EMR Launch

During the Amazon EMR Launch implementation, the AWS Professional Services team provides an Amazon EMR platform foundation architecture that enables Hadoop workloads to execute with compute decoupled from storage on an Amazon S3 data lake using infrastructure as code. The AWS Professional Services team helps in the following areas:

- Deploying Amazon EMR using AWS CloudFormation templates.
- Guidance in designing and implementing data security requirements on Amazon EMR.
- Ensuring EMR can access defined S3 buckets.
- Setting up IAM roles, security groups, and Active Directory using CloudFormation templates.
- Setting up AWS Glue or Amazon RDS for external Apache Hive Metastore for Amazon EMR.
- Setting up Amazon EMR cluster and performing tuning one workload

The goals of the Amazon EMR Launch phase include:

- A defined workload automated and tuned on EMR
- Automated EMR Infrastructure as Code to support transient clusters and end-user access for development and research.
- Documented performance and testing results.
- Refined AWS Services Cost Estimate *(Platform Spend)*

For more information on work effort, timeline and cost, contact your Sales team or click here to connect directly with AWS Professional Services leadership.

AWS Partners

In addition to AWS Professional Services team, AWS has a robust network of firms included in the AWS Service Delivery Program. The AWS Service Delivery Program is a validation program that identifies and endorses APN Partners with customer experience and a deep understanding of specific AWS services. Firms that meet the criteria are listed by geography and can be found on the Amazon EMR Partner site.

AWS Support

AWS Support plans are designed to give you the right mix of tools and access to expertise so that you can be successful with AWS while optimizing cost, performance, and managing risk.

Basic Support for Amazon EMR

Basic Support is included for all AWS customers and includes:

- Customer Service and Communities - 24x7 access to customer service, documentation, whitepapers, and support forums.
- AWS Trusted Advisor - Access to the seven core Trusted Advisor checks and guidance to provision your resources following best practices to increase performance and improve security.
- AWS Personal Health Dashboard - A personalized view of the health of AWS services, and alerts when your resources are impacted.

AWS Premium Support for Amazon EMR

For applications and configuration found within our documentation, AWS Support provides the necessary protocols to resolve your issue. However, since an Amazon EMR cluster can be infinitely customized, applications and configurations that are outside of our documentation, including changes on Custom AMI features (previously called AMIs for EMRs Bring Your Own AMI feature), are supported at best effort as long as the AMI has not been deprecated.[8] The Amazon EMR team recommends engaging your Technical Account Manager to find a supported solution or to minimize the risk associated with a particular customization.

AWS Premium Support includes:

- Answering “how to” questions about AWS services and features
- Providing best practices to help you successfully integrate, deploy, and manage applications in the AWS Cloud.
- Troubleshooting API and AWS SDK issues
- Troubleshooting operational or systemic problems with AWS resources

- Troubleshooting issues with the AWS Management Console or other AWS tools
- Debugging problems detected by EC2 health checks
- Troubleshooting third-party applications such as OS, web servers, email, VPN, databases, and storage configuration

AWS Premium Support does not include:

- Code development
- Debugging custom software
- Performing system administration tasks
- Tuning queries

Third-Party and Marketplace Support

Third-party products, including partner solutions, are typically outside of any AWS Support team scope. Although we may be able to help on a best effort basis, we suggest you source guidance from the AWS Partner Network or directly from the company offering the product.

Several AWS Marketplace products offer support, either directly in Marketplace or through the originating vendor website. For more information, see New – Product Support Connection for AWS Marketplace Customers.

Contributors

Contributors to this document include:

- Mert Hocanin, Principal Big Data Architect, AWS
- Tanzir Musabbir, Sr. Big Data Specialist SA, AWS
- Nikki Rouda, Principal Product Marketing Manager, AWS
- Radhika Ravirala, Principal Big Data Specialist SA, AWS
- Sreekanth Munigati, Sr. Big Data Specialist SA, AWS
- Fei Lang, Sr. Big Data Architect, AWS
- Veena Vasudevan, Big Data Architect, AWS
- Imtiaz Sayed, Analytics Tech Lead, AWS
- Bruno Faria, Principal Big Data Specialist SA, AWS
- Rahul Bhartia, Sr. Manager, AWS
- Rajesh Ramchander, Sr. Big Data Consultant, AWS

- Damon Cortesi, Big Data Architect, AWS
- Matthew Liem, Sr. Big Data Architect, AWS
- Tony Nguyen, Principal Consultant, AWS Professional Services
- Drew Alexander, Principal Business Development, AWS
- Jason Berkowitz, Sr. Practice Manager – Data Lake, AWS
- Saurabh Bhutyani, Sr. Big Data Specialist SA, AWS
- Amir Basirat, Big Data Specialist SA, AWS
- Darren Yohan, Cloud Infrastructure Architect, AWS
- Kalyan Janaki, Sr. Technical Account Manager, AWS
- Chinmayi Narasimhadevara, Solutions Architect, AWS
- Rajarao Vijjapu, Security Data Architect, AWS
- Neal Rothleder, Practice Manager, AWS

Additional Resources

For additional information, see these resources:

- Amazon EMR webpage
- About Amazon EMR Releases
- AWS Big Data Blog

Blog Posts

- A Hybrid Cloud Architecture Ah-Ha! Moment
- 4 Dos and Don’ts When Using the Cloud to Experiment
- Top Performance Tuning Tips for Amazon Athena
- Untangling Hadoop YARN
- Orchestrate Apache Spark applications using AWS Step Functions and Apache Livy
- Orchestrate multiple ETL jobs using AWS Step Functions and AWS Lambda
- Top 10 Performance Tuning Tips for Amazon Athena
- Easily manage table metadata for Presto running on Amazon EMR using the AWS Glue Data Catalog

Whitepapers & Guides

- Amazon EMR Release Guide
- Amazon EMR Management Guide
- Getting Started with Amazon EMR
- Building a Multitenant Storage Model on AWS
- Big Data Analytics Options on AWS

Analysts Reports

- The Economic Benefits of Migrating Apache Spark and Hadoop to Amazon EMR
- The Forrester Wave: Cloud Hadoop/Spark Platforms, Q1 2019

Media

- AWS re:Invent Videos
- Amazon EMR on YouTube

Document Revisions

Date	Description
August 2022	Updated for AWS DataSync
December 2020	Updated for AWS Lake Formation Metadata Authorization, AWS Outposts, Amazon EMR Notebooks, Amazon EMR Migration Program, Incremental Data Processing.
August 2019	Updated title and introduction.
June 2019	First publication.

Appendix A: Questionnaire for Requirements Gathering

For migrating from on-premises to Amazon EMR, Amazon Athena, and AWS Glue, this questionnaire helps you take inventory of the current architecture and the possible requirements for migration.

Current Cluster Setup

- What does the current cluster look like?
  - How many nodes?
  - How much data is stored on the cluster?
- Which distribution of Apache Hadoop and/or Apache Spark are you running on?

Cluster Use

- How much of the cluster is being used on average, during peak, during low times?
  - How many users, what percentage of CPU and memory?
  - Where are users located? One time zone or spread globally?
- How much of the data is being accessed regularly?
- How much new data is added on a monthly basis?
- What kind of data formats are the source, intermediate, and final outputs?
- Are workloads segregated in any manner? (i.e. with queues or schedulers)

Maintenance

- How is the cluster being administrated right now?
- How are upgrades being done?
- Are there separate development and production clusters?
- Is there a backup cluster or data backup procedure?

Use Cases

Batch Jobs

- How many jobs per day?
- What is the average time they run?

- When do they usually run?
- What types of work do the batch jobs run? (i.e. machine learning, aggregation, data format translation, and so on). (Optimize for machine types.)
- What frameworks or languages do the batch jobs use? (i.e. Apache Spark, HiveQL, and so on)
- Are there service level agreements (SLAs) to downstream consumers?
- How are jobs promoted to the batch environment?
- How are jobs submitted to the cluster?
- Are jobs designed to be idempotent?

Interactive Use Cases

- Is there interactive access to a cluster?
- Who is using the clusters?
- How are the clusters secured?
- How many people are using them?
- What tools or apps do people use to connect?
- If using Spark, how are Spark jobs deployed and submitted?

Amazon Athena Use Cases

- What is the query load? Is Amazon Athena more appropriate for your use case?

AWS Glue Use Cases

- What load is expected on AWS Glue?
- Is there an existing Hive Data Catalog? Can the data catalog be migrated to AWS Glue?

Security Requirements

- Are you using Kerberos?
- Are you using a directory service? Which one?
- Are there fine-grained access control requirements? How is read/write access restricted?
- Are users allowed to create their own tables? How do you track data lineage?

TCO Considerations

Growth expectations over time

- How much will the data grow over time?

- How much will compute needs grow over time?
- Are there network transfer costs to be aware of? (Include costs both during migration and after migration, for example, cross-region replication.)
- Is there a target date for decommissioning the existing environment?
- What is the cost of maintaining the existing infrastructure during transition?

Appendix B: EMR Kerberos Workflow

MIT's open source version of KDC is installed on Amazon EMR. The KDC uses default settings. For a detailed explanation on its configuration and operation, see MIT's Kerberos documentation.

The subsections in this appendix show how users interact with a Kerberos enabled Amazon EMR cluster and flow of messages.

EMR Kerberos Cluster Startup Flow for KDC with One-Way Trust

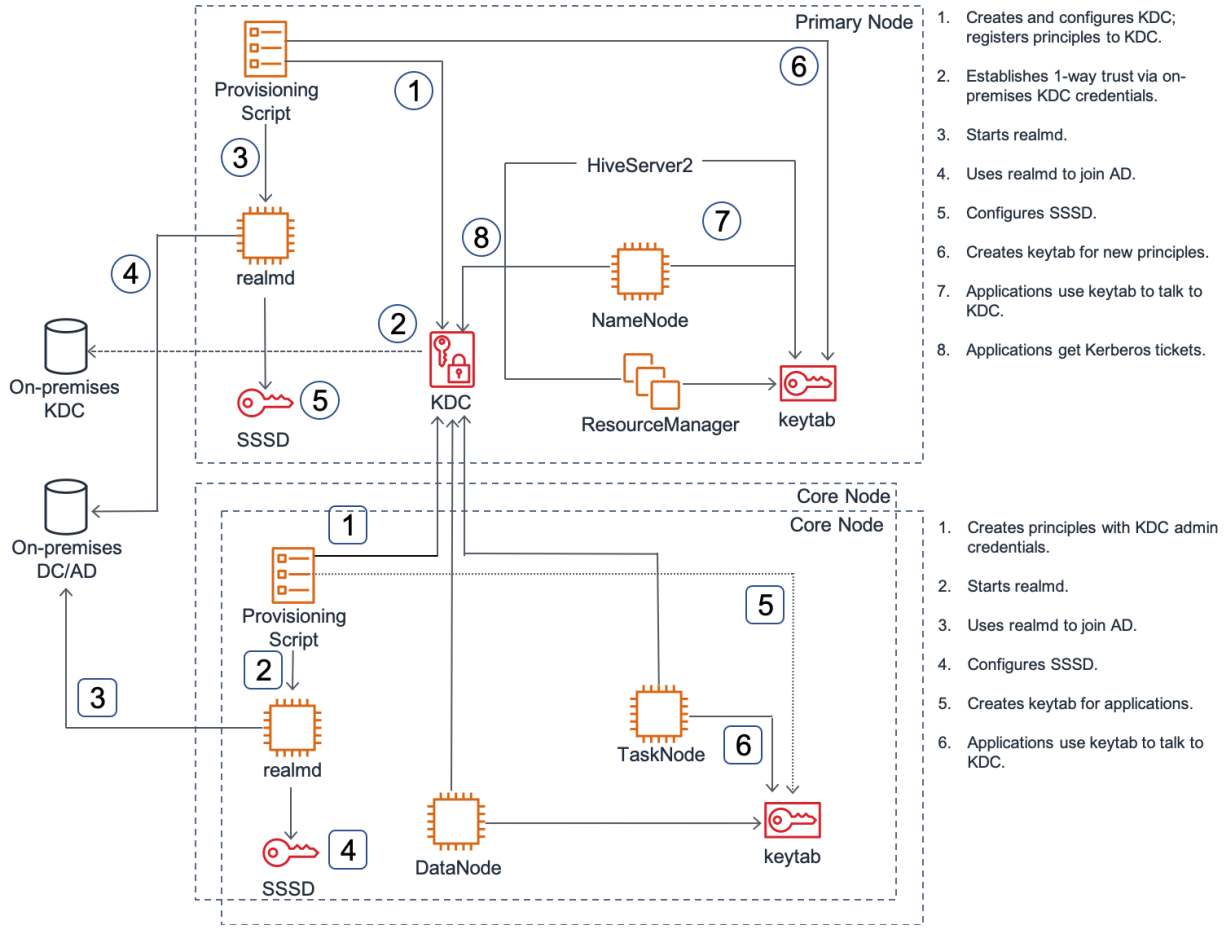


Figure 50: EMR Kerberos Cluster Startup Flow for KDC with One-Way Trust

In this flow, each node has a provisioning script that is responsible for:

- (Primary node only) Creates the KDC and configures it for 1-way trust.
- (All nodes) Start realmd to join Active Directory, which then configures SSSD to handle user and group mapping requests and create keytab.

- (All nodes) Create application principles and keytabs for each application and sub-application.
- When a new node comes up in the cluster, its provisioning script performs the same operations: creates principles within the KDC on the master node for all the applications running locally, creates the keytab file, joins the node to Active Directory (if configured), starts the applications.

EMR Kerberos Flow Through Hue Access

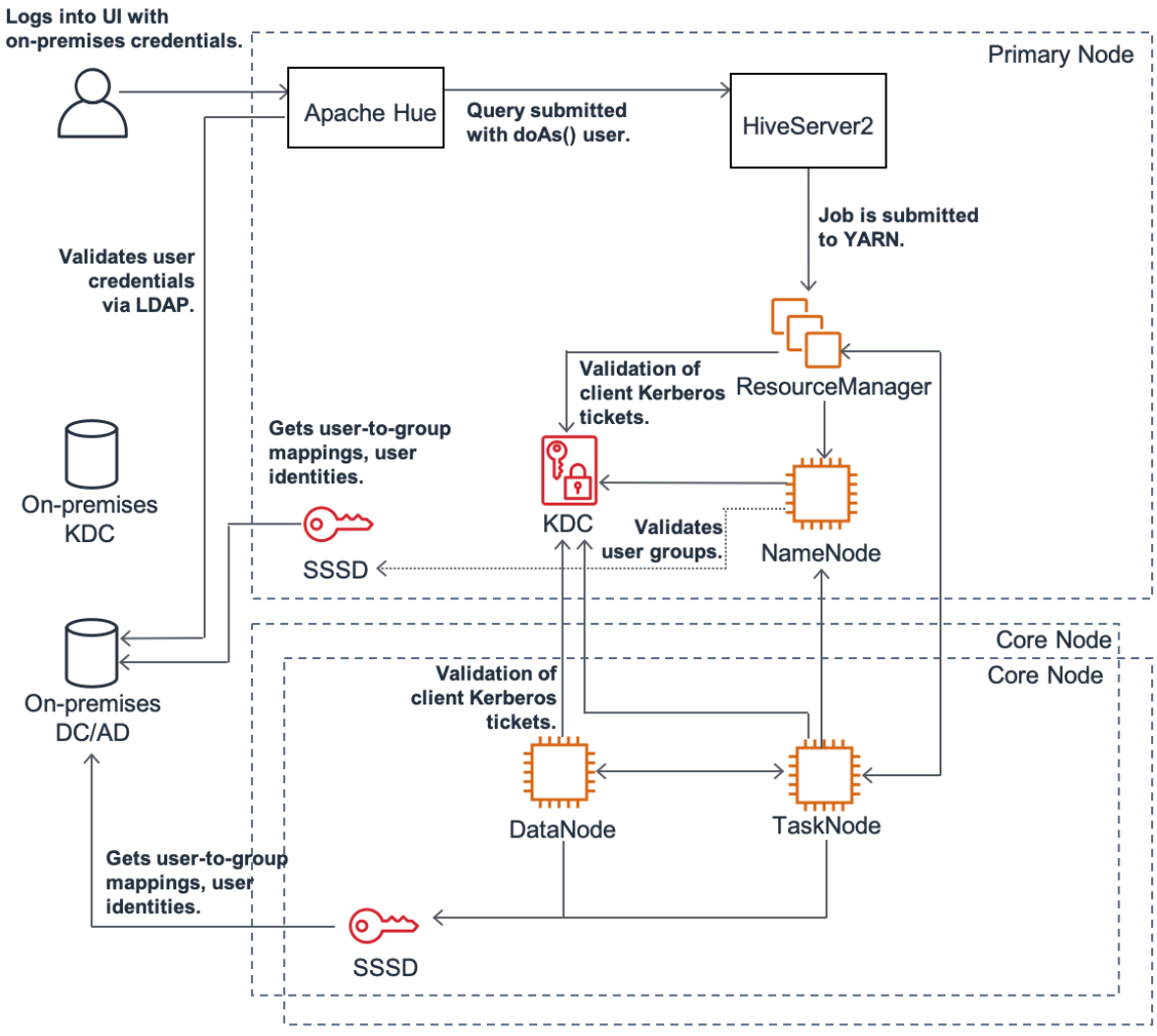


Figure 51: EMR Kerberos Flow through Hue access

1. Users log into Hue (or Zeppelin) using their on-premises credentials.
2. Hue authenticates those credentials using LDAP(S) with the on-premises Active Directory.
3. Once authenticated, the user can submit Hive queries.

4. Hue submits those queries and tells HiveServer2 to run the job as the user (i.e. impersonation.)
5. HiveServer2 submits the job to resource manager for processing.
6. During the execution of the job, Hadoop authenticates and authorizes the user by using SSSD to verify the user account on the local node.

EMR Kerberos Flow for Directly Interacting with HiveServer2

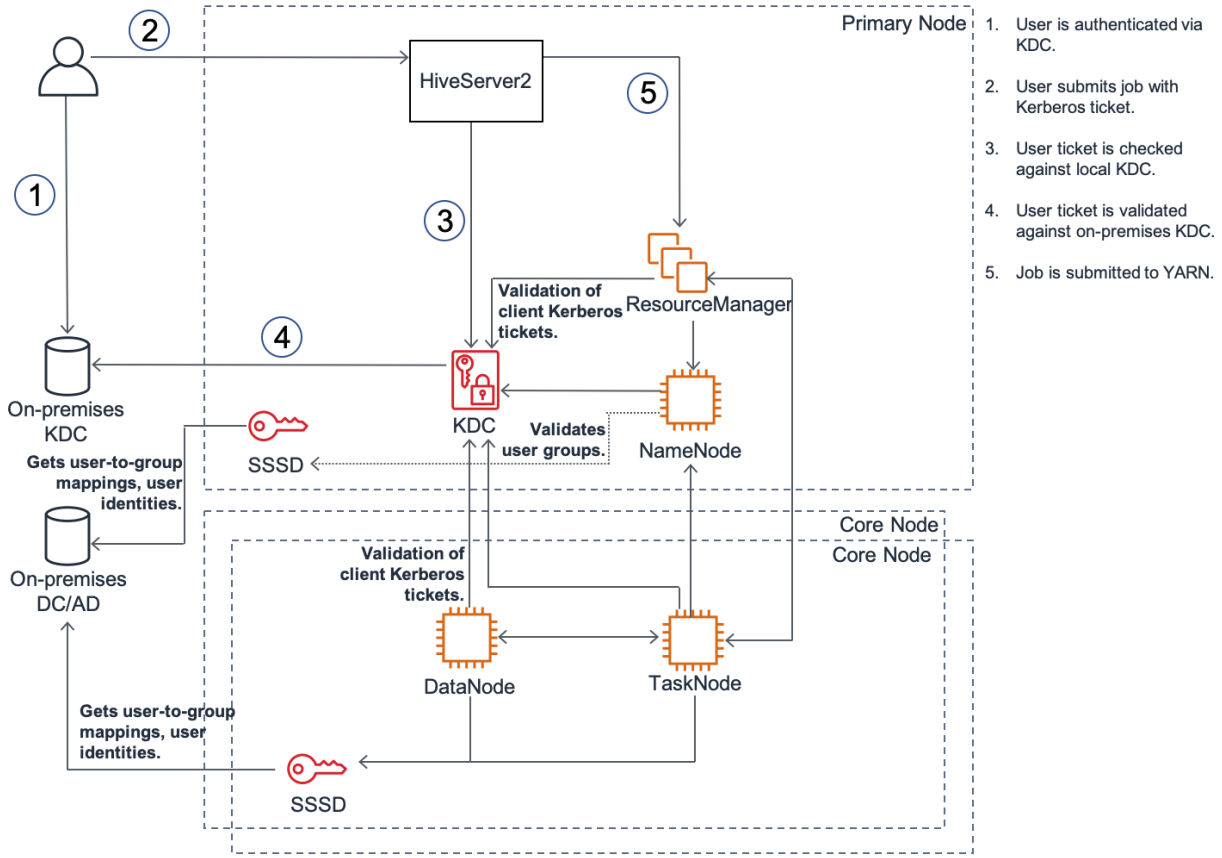


Figure 52: EMR Kerberos flow for directly interacting with HiveServer2

1. User authenticates using credentials for the local KDC and receives a Kerberos ticket.
2. User submits a query and the Kerberos ticket to HiveServer2.
3. HiveServer2 requests the local KDC to validate the Kerberos ticket.
4. The local KDC requests the on-premises KDC to validate the Kerberos ticket.
5. HiveServer2 submits the job to Resource Manager for processing as the user.
6. During the execution of the job, Hadoop authenticates and authorizes the user by using SSSD to verify the user account on the local node.

EMR Kerberos Cluster Startup Flow

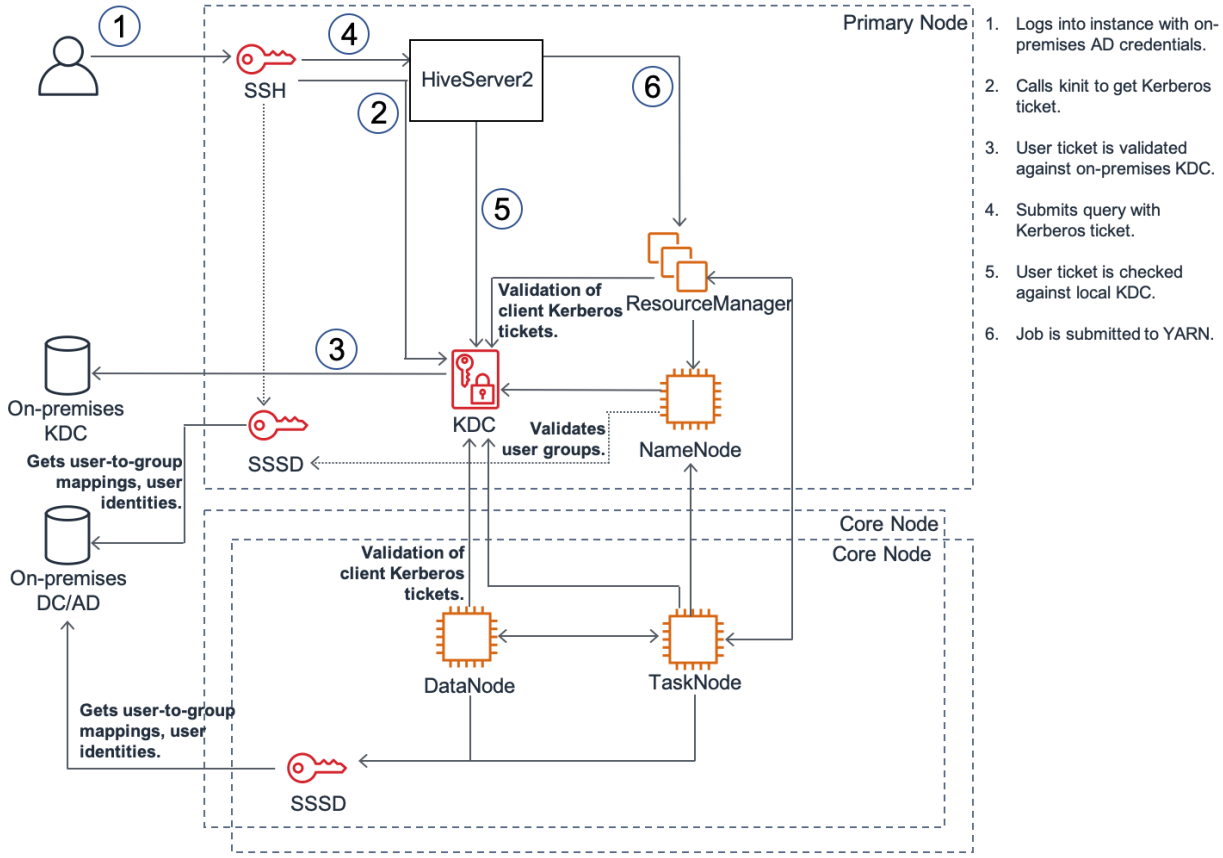


Figure 53: EMR Kerberos cluster startup flow

1. User accesses the primary node through SSH and authenticates with user credentials.
2. If the user needs a Kerberos ticket, the user must kinit (kinit -k -t <keytab> <principal>) to get a Kerberos ticket using their credentials from the local KDC.
3. The local KDC uses on-premises KDC to authenticate the user and return a Kerberos ticket.
4. User submits the query and the Kerberos Ticket to HiveServer2.
5. HiveServer2 requests that the local KDC validate the ticket.
6. The local KDC requests the on-premises KDC to validate the ticket.
7. HiveServer2 submits the job to Resource Manager for processing as the user.
8. During job execution, Hadoop uses SSSD to authenticate and authorize the user account on the local node.

Appendix C: Sample LDAP Configurations

Example LDAP Configuration for Hadoop Group Mapping

Below shows a sample Amazon EMR configuration file to set up Hadoop Group Mapping to use LDAP directly. Use this setup for clusters without Kerberos.

```
[

{
  "classification":"core-site",
  "properties":{
     "hadoop.security.group.mapping.ldap.search.attr.member":"member",
     "hadoop.security.group.mapping.ldap.search.filter.user":"(objectclass=*)",
     "hadoop.security.group.mapping.ldap.search.attr.group.name":"cn",
     "hadoop.security.group.mapping.ldap.base":"dc=corp,dc=emr,dc=local",
     "hadoop.security.group.mapping":"org.apache.hadoop.security.LdapGroupsMapping",
     "hadoop.security.group.mapping.ldap.url":"ldap://172.31.93.167",
     "hadoop.security.group.mapping.ldap.bind.password":"Bind@User123",
     "hadoop.security.group.mapping.ldap.bind.user":"binduser@corp.emr.local",
     "hadoop.security.group.mapping.ldap.search.filter.group":"(objectclass=*)"
  },
  "configurations":[
  ]
 }
]
```

Example LDAP Configuration for Hue

```
[
 {
    "classification":"hue-ini",
    "properties":{
    },
    "configurations":[
       {
          "classification":"desktop",
          "properties": { },
            "configurations":[
               {
                  "classification":"auth",
                  "properties":{
                     "backend":"desktop.auth.backend.LdapBackend"
                  },
                  "configurations": []
               },
```

```
                {
                   "classification":"ldap",
                   "properties":{
                      "bind_dn":"binduser@corp.emr.local",
                      "trace_level":"0",
                      "search_bind_authentication":"false",
                      "debug":"true",
                      "base_dn":"dc=corp,dc=emr,dc=local",
                      "bind_password":"Bind@User123",
                      "ignore_username_case":"true",
                      "create_users_on_login":"true",
                      "ldap_username_pattern":"uid=<username>,cn=users,dc=corp,dc=emr,dc=
local",
                      "force_username_lowercase":"true",
                      "ldap_url":"ldap://172.31.93.167",
                      "nt_domain":"corp.emr.local"
                   },
                   "configurations":[
                      {
                         "classification":"groups",
                         "properties":{
                            "group_filter":"objectclass=*",
                            "group_name_attr":"cn"
                         },
                         "configurations": []                        },
                      {
                         "classification":"users",
                         "properties":{
                            "user_name_attr":"sAMAccountName",
                            "user_filter":"objectclass=*"
                         },
                         "configurations":[]
                      }
                   ]
                }
             ]
          }
       ]
    }
]
```

Appendix D: Data Catalog Migration FAQs

What are some of the limitations of using an AWS Glue Data Catalog over a generic Hive metastore?

Column statistics, Hive authorizations, and Hive constraints are not supported on AWS Glue Data Catalog. To see a list of AWS Glue Data Catalog constraints, see *Considerations when Using AWS Glue Catalog* in Using the AWS Glue Data Catalog as the Metastore for Hive.

What types of security features are available for an AWS Glue Data Catalog?

You can enable encryption on an AWS Glue Data Catalog, and access to AWS Glue actions are configurable through IAM policies. The default Amazon EMR EC2 role (EMR_EC2_DefaultRole) allows the required AWS Glue actions. However, if you specify a custom EC2 instance profile and permissions when you create a cluster, ensure that the appropriate AWS Glue actions are allowed. For a list of available Glue IAM policies, see AWS Glue API Permissions: Actions and Resources Reference.

Can multiple Amazon EMR clusters use a single AWS Glue Data Catalog?

Yes, an AWS Glue Data Catalog can be used by one-to-many Amazon EMR clusters, as well as Amazon Athena and Amazon Redshift.

Can an on-premises Hadoop cluster use AWS Glue Data Catalog?

No, AWS Glue Data Catalog libraries are not yet open source, and AWS Glue Data Catalog cannot be used to connect from an on-premises Hadoop cluster.

When should I use a Hive metastore on Amazon RDS over an AWS Glue Data Catalog?

If you want full control of your Hive metastore and want to integrate with other open-source applications such as Apache Ranger and Apache Atlas, then use Hive metastore on Amazon RDS. If you are looking for a managed and serverless Hive metastore, then use AWS Glue Data Catalog.

Notes

[1] For a step-by-step guide on how to set up an LDAP server and integrate Apache Hue with it, see Using LDAP via AWS Directory Service to Access and Administer Your Hadoop Environment on the *AWS Big Data Blog*.

[2] Example customers that use Amazon S3 as their storage layer for their data lakes include NASDAQ, Zillow, Yelp, iRobot, and FINRA.

[3] For more information on these features, see Configuring Node Decommissioning Behavior.

[4] For Amazon EMR versions 5.8.0 and later, you can configure Hive to use the AWS Glue Data Catalog as its metastore. See Existing Hive Metastore to AWS Glue Data Catalog in Data Catalog Migration.

[5] Applies to Amazon EMR software version 5.20 and later.

[6] For example, FINRA migrated a 700-TB HBase environment to HBase on Amazon S3. For more information, see Low-Latency Access on Trillions of Records: FINRA's Architecture Using Apache HBase on Amazon EMR with Amazon S3.

[7] V.M. Megler, Kristin Tufte, and David Maier, *Improving Data Quality in Intelligent Transportation Systems*, https://arxiv.org/abs/1602.03100 (Feb. 9, 2016)

[8] For information on support, see Amazon EMR What's New and the Amazon EMR FAQs.