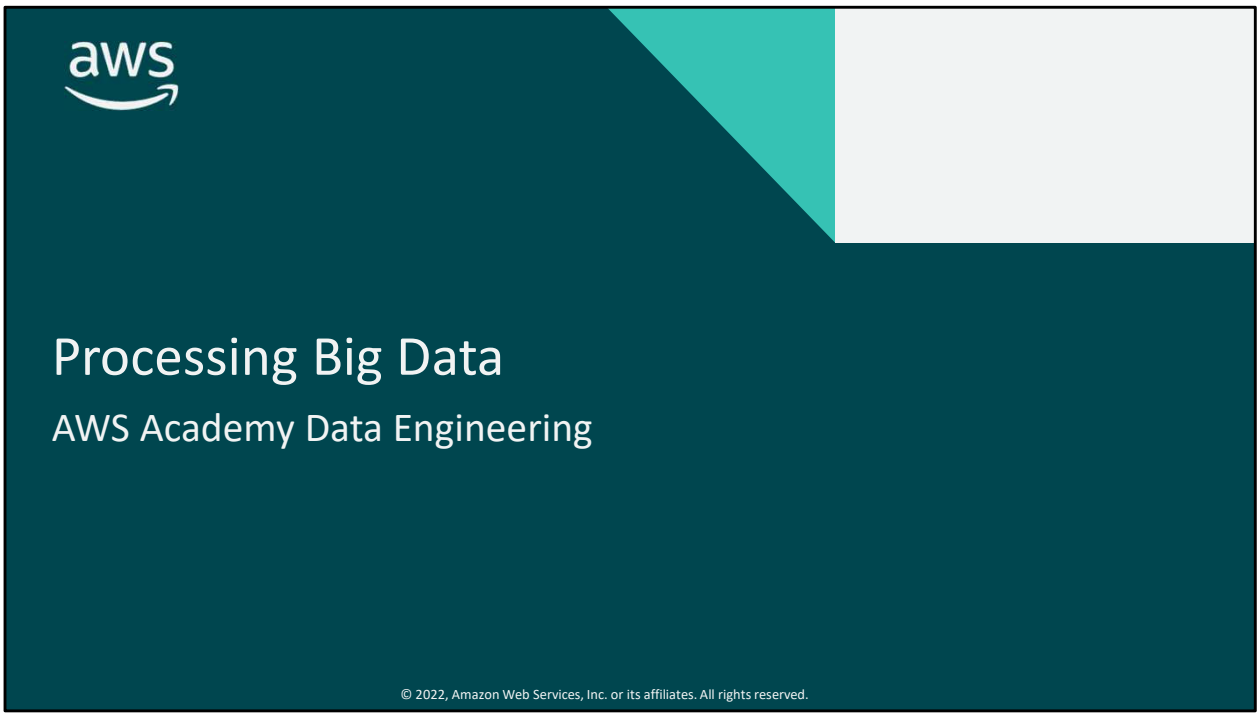
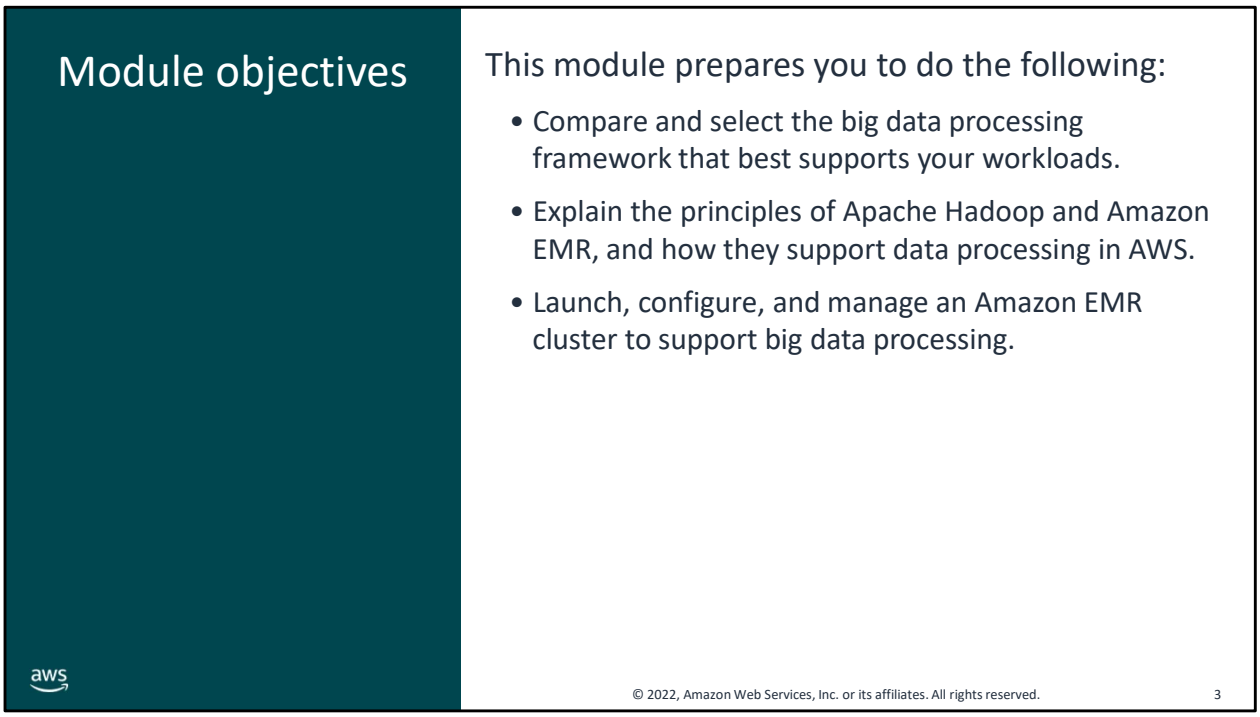


|Slide number 1
|Instructor notes
|
|Student notes
Welcome to the Processing Big Data module.

|Slide number 2
|Instructor notes
|
|Student notes
This introduction section describes the content of this module.



|Slide number 3
|Instructor notes
|
|Student notes
This module builds on the ingestion and storage elements of the big data pipeline that previous modules covered. This module highlights the processing frameworks that are integral to the AWS services that support big data processing in the AWS Cloud.

Module overview

Presentation sections

- Big data processing concepts
- Apache Hadoop
- Apache Spark
- Amazon EMR
- Managing your Amazon EMR clusters
- Apache Hudi

Labs

- Processing Logs by Using Amazon EMR
- Updating Dynamic Data in Place

Knowledge checks

- Online knowledge check
- Sample exam question

aws

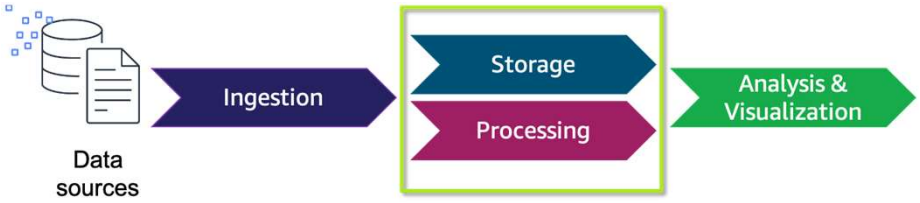
|Slide number 4
|Instructor notes
|Each module has an introduction, content sections, and a wrap-up. The wrap-up for this module contains a sample exam question for you to review with the students.
|
|Student notes
The objectives of this module are presented across multiple sections.

You will also complete hands-on labs. In the first lab, you will learn about using Amazon EMR to process logs. In the second lab, you will learn how to use Amazon Simple Storage Service (Amazon S3), Amazon Athena, AWS Glue, and Apache Hudi to facilitate in-place data updates and run queries to get data.

The module wraps up with a sample exam question and an online knowledge check that covers the presented material.

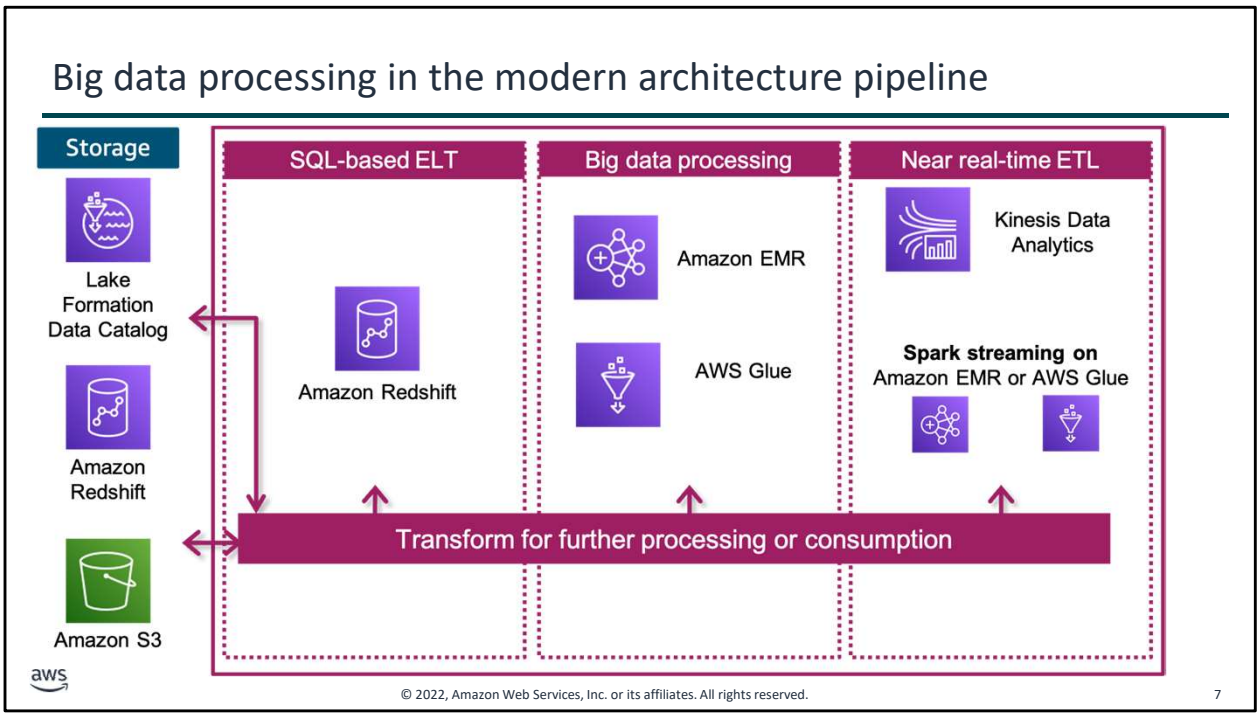
|Slide number 5
|Instructor notes
|
|Student notes
This section introduces concepts for processing big data.

Big data processing and the iterative data pipeline



|Slide number 6
|Instructor notes
|
|Student notes
Up to this point in the course, you have learned about the roles that ingestion and storage play in the pipeline. You also learned how closely related they are to the processing phase of the iterative data pipeline. Data frequently flows between the processing and storage elements. Raw data is ingested, stored, transformed, and processed. That data often returns to storage in a new format, and that formatted data is used for analysis and visualization.

In this module, you will learn about some of the common frameworks for processing big data and the services that are used to process data for your analytics and machine learning (ML) pipelines.



|Slide number 7
|Instructor notes
|
|Student notes
In the Design Principles and Patterns for Data Pipelines module, you learned that components in the data processing layer are responsible for transforming data into a consumable state. The processing layer provides purpose-built components that enable a variety of data types, velocities, and transformations. Each component can read and write data to both Amazon S3 and Amazon Redshift in the storage layer, and all can scale to high data volumes.

This architecture supports multiple ETL or ELT pipelines that perform iterative processing of data for different types of preparation and consumption. Each pipeline reads data from the storage layer, processes it using temporary storage as needed, and then writes it to the appropriate location within the storage layer. This module focuses on the big data processing aspect of this architecture.

Types of data processing

Batch data processing	Streaming data processing
• Infrequently accessed (cold) data querying • Processes input data in batches at varying intervals • Tolerates structured and unstructured data • Capable of deep analysis of big datasets • Examples: Amazon EMR, Apache Hadoop	• Frequently accessed (hot) data querying • Processes data sequentially and incrementally in near real time • Capable of processing less predictable data on a massive scale • Enables analysis of continually generated data • Examples: Amazon Kinesis Data Streams, Apache Spark Streaming

|Slide number 8
|Instructor notes
|Review the key differences between batch data processing and streaming data processing, namely the type of data they work with and whether data is processed in intervals or in near-real time.
|
|Student notes
Batch data processing
Batch data processing typically involves querying large amounts of *cold* data—data that is infrequently accessed. With batch processing, it might take hours to analyze data to get answers to business questions. For example, you might use batch processing to generate a billing report at the end of the month.

Batch processing jobs typically must scan all their new input data; therefore, they can tolerate data that has only partial or no structure. Batch processing jobs can also compute results that are derived from all the available data, so they enable deep analysis of big datasets. Additionally, you can use batch jobs to compute the same query over different datasets.

Hadoop MapReduce-based systems, such as Apache Hadoop and Amazon EMR, are examples of platforms that support batch-based data processing jobs.

Streaming data processing

Streaming data is generated continuously by thousands of data sources. These sources typically send in the data records simultaneously and in small sizes (order of kilobytes). Streaming data includes a wide variety of data, such as log files that are generated when customers use your mobile or web applications, ecommerce purchases, in-game player activity, and information from social networks. Streaming data can also come from financial trading floors, geospatial services, and telemetry from connected devices or instrumentation in data centers.

This data needs to be processed sequentially and incrementally on a record-by-record basis or over sliding time windows. The data is used for a wide variety of analytics, including correlations, aggregations, filtering, and sampling. Information from such analysis gives companies visibility into many aspects of their business and customer activity. For example, streaming data can provide information about service usage (for metering or billing), server activity, website clicks, and geo-location of devices, people, and physical goods. This information can help companies to respond promptly to emerging situations. For example, a business could track changes in public sentiment on their brands and products by continuously analyzing social media streams and respond in a timely fashion as the necessity arises.

Streaming data processing is beneficial in most scenarios where new dynamic data is generated on a continual basis. Stream processing applies to most industry segments and big data use cases. Companies generally begin with simple applications, such as collecting system logs, and rudimentary processing, such as rolling min-max computations. Then, these applications evolve to more sophisticated near real-time processing. Initially, applications might process data streams to produce simple reports and perform simple actions in response. For example, alarms could be emitted when key measures exceed certain thresholds. Eventually, those applications perform more sophisticated forms of data analysis, such as applying ML algorithms, and extract deeper insights from the data.

Frameworks that support batch and stream processing

Batch Processing	Stream Processing	
Apache Spark	Amazon Kinesis	
Apache Hadoop MapReduce	Apache Spark Streaming	AWS Lambda
Apache Hive	Apache Hive	Apache Flink
Apache Pig	Apache Pig	Apache Storm

|Slide number 9
|Instructor notes
|
|Student notes
When you launch your cluster, you choose the frameworks and applications to install that best fit your data processing needs. *Frameworks* are the data processing engines that are used to process and analyze your data. You can select frameworks based on whether your data processing is batch, interactive, in-memory, streaming, or something else. Hadoop and Spark are the two most common frameworks that amazon EMR uses. You can use *applications* to create processing workloads with higher level languages, use ML algorithms, make stream processing applications, and build data warehouses.

Many frameworks are available for processing big data, and new frameworks and enhancements to existing frameworks are released with increasing frequency. A single framework will probably not meet all of your big data analytics needs, so it's common to use combinations of processing frameworks.

The table on the slide displays a number of popular frameworks for processing big data. The frameworks are organized by the use case type (batch or stream processing) that each framework is best suited to support. Some of these frameworks have been previously discussed in this course, while others will be covered in this and subsequent lessons.

Key takeaways: Big data processing concepts

- Big data processing is generally divided into two categories, batch and streaming.
- Batch data typically involves cold data, with analytics workloads that involve longer processing times.
- Streaming data involves many data sources providing data that must be processed sequentially and incrementally.
- Batch processing and stream processing each benefit from specialized big data processing frameworks.

|Slide number 10
|Instructor notes
|
|Student notes
Here are a few key points to summarize this section.

Big data processing is generally divided into two categories based on data type—batch and streaming.

Batch data typically involves infrequently accessed cold data, and analytics workloads often involve longer processing times.

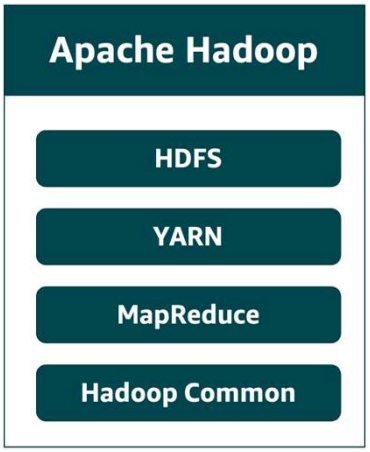
Streaming data involves numerous hot, or frequently accessed, data sources. The data must be processed in a sequential and incremental fashion.

Batch processing and stream processing each benefit from specialized big data processing frameworks that are tailored to their processing needs.

|Slide number 11
|Instructor notes
|
|Student notes
This section introduces the Apache Hadoop processing framework.

Apache Hadoop characteristics

- Is an open-source, distributed processing framework
- Enables distributed storage and processing for large amounts of data
- Maps tasks to nodes within clusters of servers
- Hadoop components: Hadoop Distributed File System (HDFS), YARN, MapReduce, Hadoop Common
- Clusters consist of main nodes and worker nodes



|Slide number 12
|Instructor notes
|
|Student notes
Hadoop is an open-source framework that uses a distributed processing architecture. Hadoop maps tasks to a cluster of commodity servers for processing. Hadoop consists of four main components: Hadoop Distributed File System (HDFS); Yet Another Resource Negotiator (YARN); MapReduce; and Hadoop Common. Hadoop clusters consist of main nodes and worker nodes. In this format, main nodes are responsible for orchestrating jobs, while worker nodes are responsible for processing those jobs.

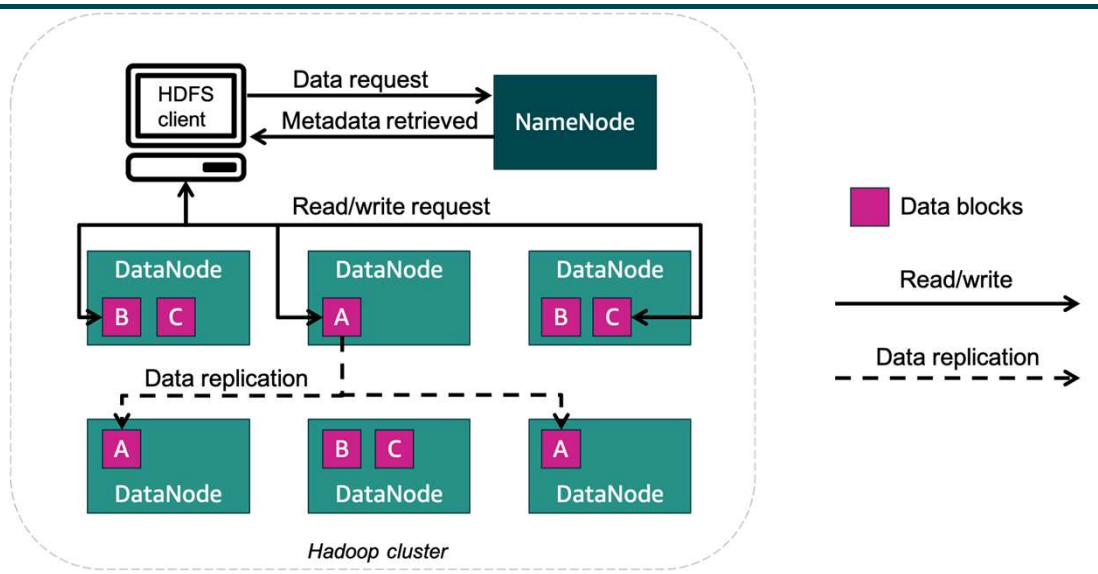
Benefits of the Hadoop framework
With Hadoop, you can store as much data as you like. You aren't required to preprocess data prior to storing it; therefore, the framework provides increased flexibility. Hadoop also has a high degree of fault tolerance, thanks to node failover. If a node fails within a cluster, its tasks are redistributed throughout the other nodes within the cluster. Data loss is likewise prevented by having multiple copies of the same data stored throughout the cluster.

The Hadoop infrastructure is scalable—as data processing and computing requirements grow, you can easily add nodes to support that growth in volume and velocity. Because it is an open-source framework, Hadoop is free and capable of running on inexpensive hardware. Finally, you can use Hadoop to process structured, unstructured, and semistructured data. You can transform the data into a variety of other formats to integrate with your existing datasets, and you can store the data with or without a schema.

Challenges of the Hadoop framework

The framework does have a few challenges. Because of its open-source nature, stability issues can arise as the framework is updated. You can avoid this challenge by running only the latest stable version. Hadoop also makes heavy use of the Java programming language—a commonly targeted language that can leave the framework open to vulnerabilities. Lastly, security concerns are inherent to Hadoop. Lack of authentication and encryption can lead to questions about the veracity of the data.

Hadoop Distributed File System (HDFS)



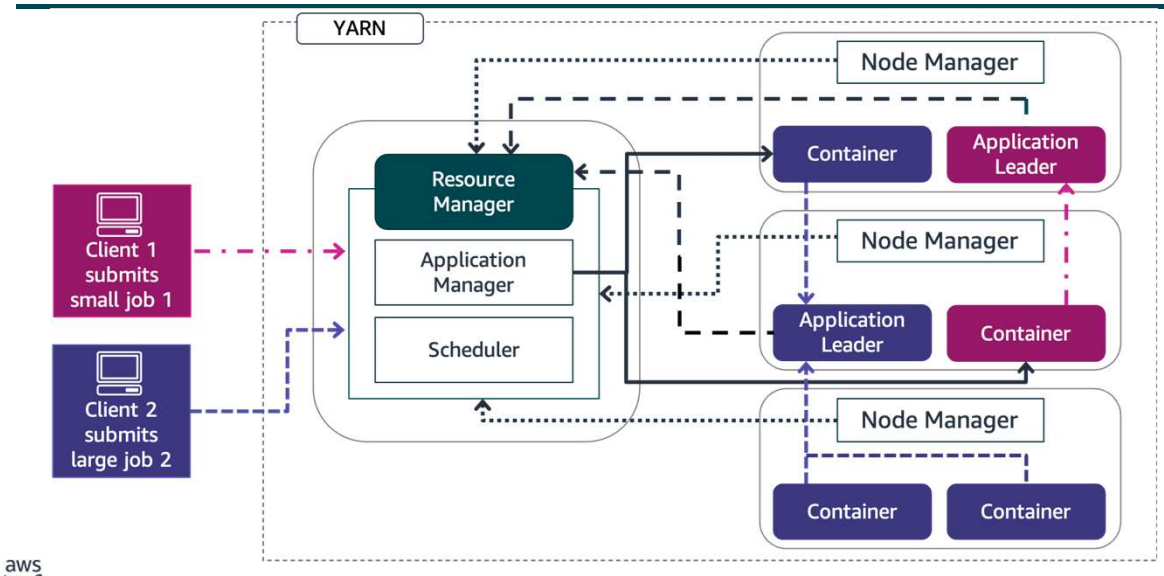
|Slide number 13
|Instructor notes
|
|Student notes

The Hadoop Distributed File System, or HDFS, is the distributed file system of the Hadoop framework. You can use HDFS to store huge amounts of data for further processing. HDFS and the Hadoop cluster use a hierarchical node architecture, which consists of a single NameNode that manages multiple DataNodes across clusters.

HDFS is typically used by applications that have large datasets, so it's designed to provide high aggregate data bandwidth. To support data velocity while storing data, HDFS splits the data into small blocks—called data blocks—and stores those blocks across several nodes of the cluster. In the example on this slide, a large data file has been split into three data blocks.

Each block resides in a different data node when possible. To avoid losing data if a cluster node fails, HDFS replicates each block several times across different nodes. This enables a high degree of fault tolerance within HDFS. The number of times that a block is replicated is called the *replication factor*, and it's a configurable setting. The cluster's NameNode catalogs the metadata about each file that is stored in the cluster. The cluster DataNodes store the data blocks for each file that has been saved to HDFS.

Yet Another Resource Negotiator (YARN)



|Slide number 14
|Instructor notes
|
|Student notes

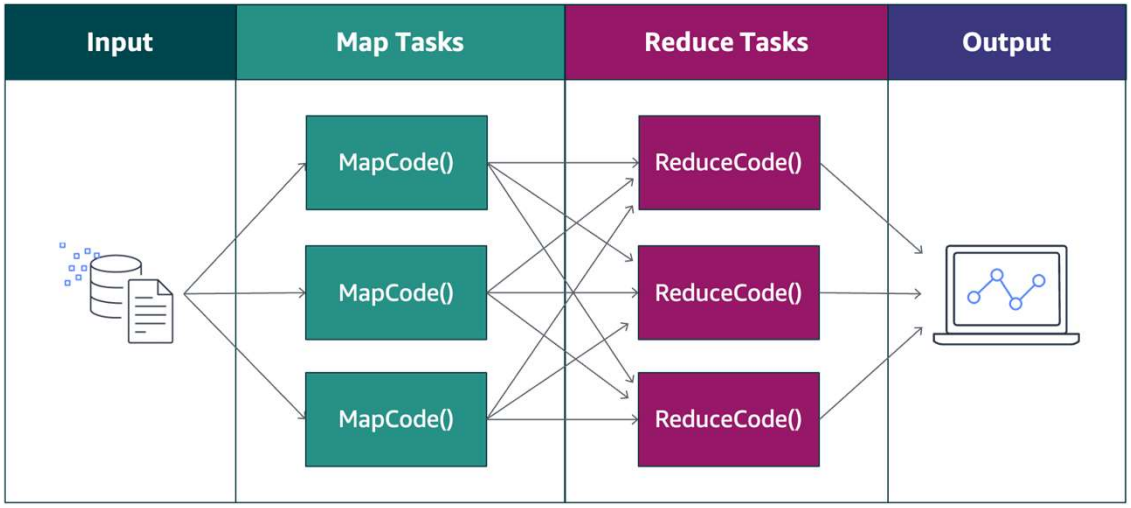
Yet Another Resource Negotiator—also called YARN—is a large-scale, distributed operating system that is used with Hadoop. YARN makes the data that is stored in HDFS accessible for different processing. YARN dynamically coordinates the use of available Hadoop resources to schedule and perform that processing.

YARN has a few primary components:

- **Resource Manager:** Controls the use of resources within the Hadoop cluster, and manages the containers that are launched on each cluster node. The Resource Manager has two main components:
    - **Scheduler:** Allocates resources to the running applications, based on the resource requirements of those applications.
    - **Application Manager:** Accepts job submissions, negotiates the first container to run the Application Leader, and provides the service to restart the Application Leader on failure.

- **Node Manager:** Controls the use of resources within a single Hadoop cluster node, and monitors the containers that are launched on that cluster node.
- **Application Leader:** Works with the Resource Manager and Node Manager to acquire cluster resources for processing tasks, before running and monitoring those tasks.
- **Containers:** Collections of cluster resources, such as memory and compute, that are allocated from a single cluster node to perform assigned processing activities.

Hadoop MapReduce

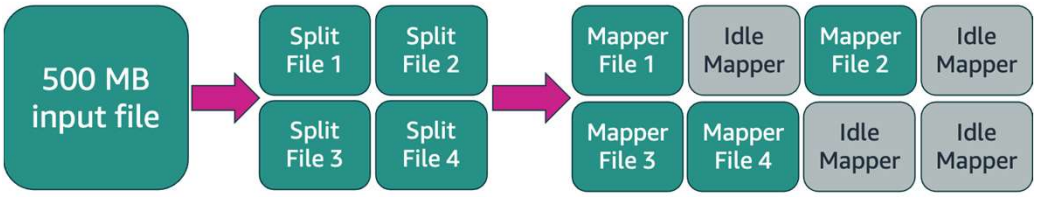


|Slide number 15
|Instructor notes
|
|Student notes
MapReduce is a framework for processing large datasets with a parallel, distributed algorithm on a cluster. Hadoop MapReduce simplifies writing parallel distributed applications by handling all of the logic, while you provide the Map and Reduce functions. The Map function maps data to sets of key-value pairs called *intermediate results*. The Reduce function combines the intermediate results, applies additional algorithms, and produces the final output. Multiple frameworks are available for MapReduce, such as Hive, which automatically generates Map and Reduce programs. The MapReduce framework is at the core of Hadoop and provides massive scalability across enormous numbers of Hadoop clusters. The framework is designed for fault tolerance, with each worker node periodically reporting its status to a leader node. The leader node can redistribute work from a cluster that doesn't respond positively.

When running a big data job, the process begins as MapReduce splits the job into discrete tasks so that the tasks can run in parallel. Next, the mapper phase maps data to key-value pairs (for example, the number of occurrences of each word on a data block). As soon as the mapper phase is finished, the next step is to shuffle and sort the data. During this step, for example, similar words are shuffled, sorted, and grouped together. The reduce phase counts the number of occurrences of words in the different groups and generates the output file.

Processing data with Hadoop MapReduce

A single file from Amazon S3 is split into four parallel HTTP requests

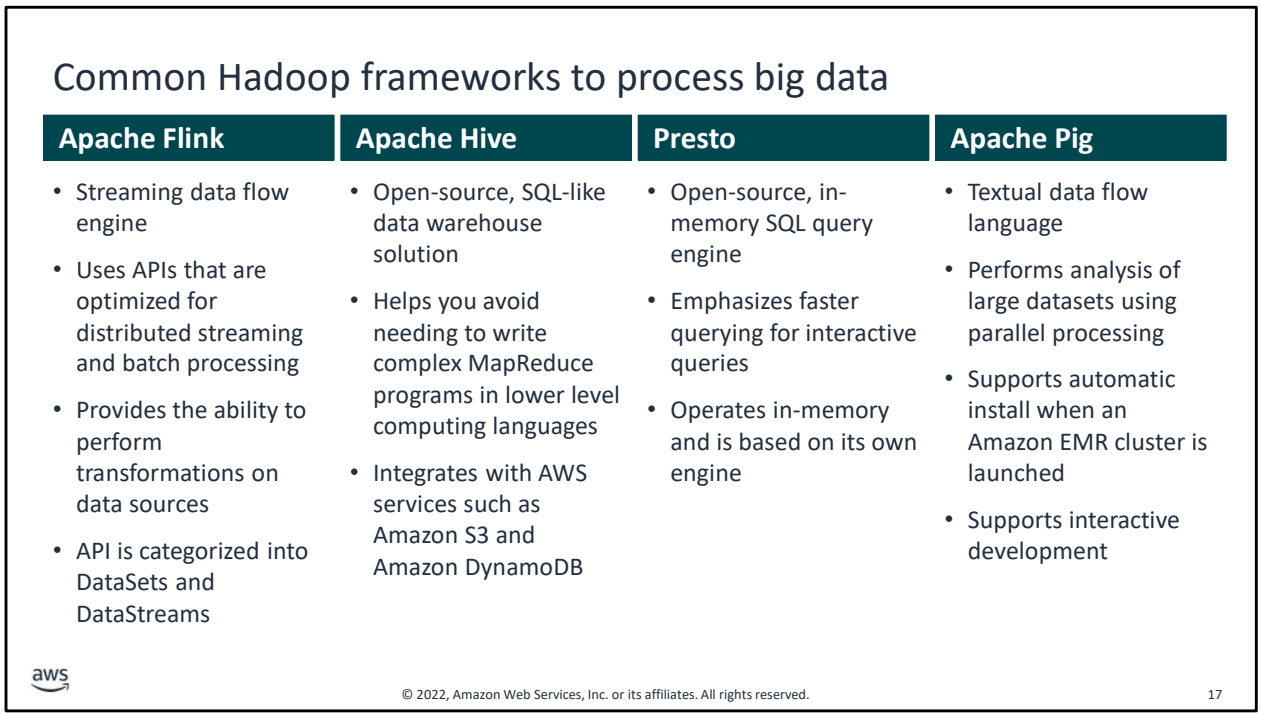


The Hadoop default split results in four files, using four of the eight available mappers

|Slide number 16
|Instructor notes
|
|Student notes

If you use HDFS to store your data, Hadoop automatically splits your data when it is stored in the HDFS cluster nodes. However, if you use Amazon Simple Storage Service (Amazon S3) to store your data, Hadoop splits the data by reading your files in multiple HTTP range requests whenever a processing job is started. The split size that Hadoop uses to read data from Amazon S3 varies depending on the Amazon EMR version that is being used. (Newer versions have larger split sizes.) The split size is generally the size of an HDFS block when operating on data that is stored in HDFS. Larger numbers provide less task granularity but also put less strain on the cluster NameNode.

In the example on the slide, the default split size has been set to 134,217,728 bytes (128 MB). Eight mapper processes are available to process the data. Big data processing needs to maintain pace with the volume and velocity of data, so in this instance the 500 MB input file will be split into four smaller files. Each of those files will use an available mapper for processing in parallel.



|Slide number 17
|Instructor notes
|
|Student notes
Let's take a look at some of the more common frameworks that work with Hadoop to process batch data, streaming data, or both.

Apache Flink
Apache Flink is a streaming data flow engine that builds real-time stream processing for big data applications. Flink performs transformations on data sources such as Amazon Kinesis Data Streams and Apache Cassandra databases. Flink uses a data streaming runtime engine, and uses pipeline parallelism to perform operations on the results of previous data transformation steps concurrently, in real time. The Flink runtime handles exchanging data between transformation pipelines and uses APIs that are optimized for distributed streaming and batch processing. The engine provides both batch and streaming APIs, and has some SQL support for these stream and batch datasets. Most of Flink's API actions are similar to the transformations on distributed object collections in Apache Hadoop or Apache Spark.

Flink's API is categorized into DataSets and DataStreams. DataSets are transformations on sets or collections of distributed data, while DataStreams are transformations on streaming data, such as data from Amazon Kinesis. Currently, Amazon EMR supports Flink as a YARN application so that you can manage resources along with other applications within a cluster. Flink-on-YARN provides an easy way to submit transient Flink jobs, or you can create a long-running cluster that accepts multiple jobs and allocates resources according to the overall YARN reservation. In addition, Flink has connectors for third-party data sources, including Kinesis Data Streams, Apache Kafka, Elasticsearch, Twitter Streaming API, and Apache Cassandra.

Apache Hive

Hive is an open-source, SQL-like data warehouse solution that runs on top of Hadoop. Hive offers some limited data warehouse capabilities, such as being able to transform source data into data warehouse-style fact schemas and dimension table star schemas. By using Hive, you can avoid the complexities of writing MapReduce programs in a lower level computing language, such as Java. SQL developers and data scientists who are looking for a familiar programming model to prototype and run jobs often find Hive to be an attractive option for big data analytics.

Presto

Presto is an open-source, distributed, in-memory SQL query engine that was originally developed by Facebook. The engine is similar to Hive because they are both SQL or SQL-like query engines. However, Presto emphasizes faster querying, specifically to perform interactive queries.

Presto doesn't run on MapReduce. Instead, the engine operates in-memory and is based on its own engine. Data scientists and analysts who need quick responses to interactive data queries might want to consider using the Presto processing framework.

Apache Pig

Pig is a simple, high-level, textual data flow language that uses parallel processing to perform analysis on large datasets. Script developers who need an intuitive way to prototype and develop jobs might want to explore Pig as a possible processing framework for their work. When users perform data loading, sorting, grouping, joining, aggregation, disaggregation, and filtering operations, they use Pig Latin to interact with data that is stored in the cluster.

Key takeaways: Apache Hadoop

- Hadoop includes a distributed storage system, HDFS.
- When you store data in HDFS, Hadoop splits the data into smaller data blocks.
- MapReduce processes large datasets with a parallel, distributed algorithm on a cluster.

|Slide number 18
|Instructor notes
|
|Student notes
Here are a few key points to summarize this section.

Hadoop includes its own distributed storage system, Hadoop Distributed File System, or HDFS.

When you store data in HDFS, Hadoop splits the data into smaller data blocks by using the MapReduce process.

MapReduce processes large datasets with a parallel, distributed algorithm on a cluster. MapReduce splits jobs into discrete tasks so that the job tasks can be run in parallel.

|Slide number 19
|Instructor notes
|
|Student notes
This section introduces the Apache Spark processing framework.

Apache Spark characteristics

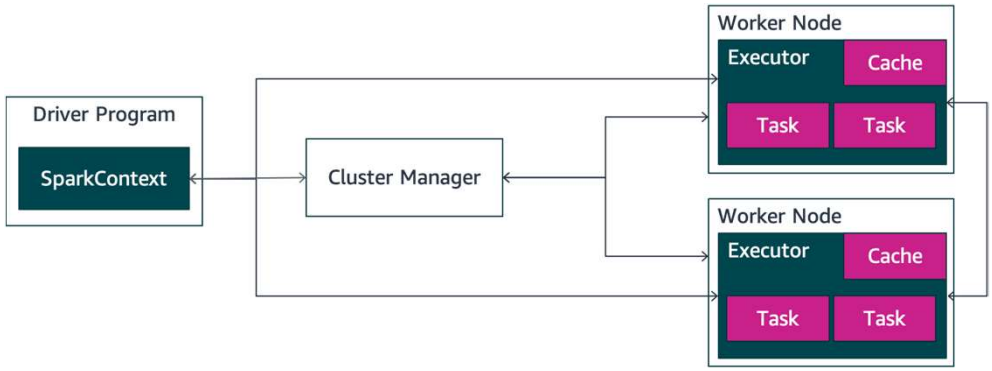
- Is an open-source, distributed processing framework
- Uses in-memory caching and optimized query processing
- Supports code reuse across multiple workloads
- Clusters consist of leader and worker nodes

|Slide number 20
|Instructor notes
|
|Student notes
Apache Spark is an open-source, distributed processing framework that was created to address the limitations of MapReduce. Spark processes data in-memory, reduces the number of steps in a job, and reuses data across multiple parallel operations. With Spark, only one step is needed, where data is read into memory, operations are performed, and the results are written back—this results in a much faster processing.

Spark is a cluster framework and programming model for processing big data workloads. When you run Spark on Amazon EMR, you can use EMRFS to directly access your data in Amazon S3. Spark supports multiple interactive query modules, such as SparkSQL.

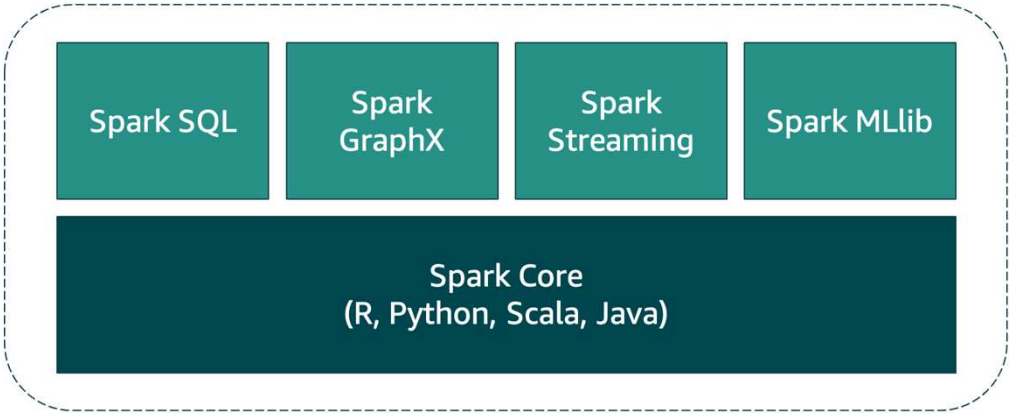
Spark also reuses data by using an in-memory cache to speed up ML algorithms that repeatedly call a function on the same dataset. Data reuse is accomplished by creating DataFrames—an abstraction over Resilient Distributed Dataset (RDD). DataFrames are a collection of objects that are cached in-memory and reused in multiple Spark operations. This dramatically lowers the latency, which makes Spark multiple times faster than MapReduce—especially for ML and interactive analytics.

Spark clusters



|Slide number 21
|Instructor notes
|
|Student notes
In Spark clusters, Spark applications run as independent sets of processes.
The *SparkContext* object that resides within the driver program coordinates these processes. Spark connects to a cluster manager, which acquires *executors* from nodes within the cluster. These executors are responsible for running the computations and storing the data in cache for your application. The SparkContext object communicates directly with the executors, sending them tasks to be processed.

Spark components



|Slide number 22
|Instructor notes
|
|Student notes
The components that make up the Spark framework are Spark Core, Spark SQL, Spark GraphX, Spark Streaming, and Spark MLlib. Let's take a closer look at each of these.

As the name implies, *Spark Core* is the foundation of the platform. It’s responsible for memory management, fault recovery, scheduling, distributing and monitoring jobs, and interacting with storage systems. Spark Core is exposed through APIs built for Java, Scala, Python, and R. These APIs hide the complexity of distributed processing behind simple, high-level operators.

Spark SQL is a distributed query engine that provides low-latency, interactive queries up to 100 times faster than MapReduce. Spark SQL includes a cost-based optimizer, columnar storage, and code generation for fast queries, while scaling to thousands of nodes.

Spark GraphX is a distributed graph processing framework that is built on top of Spark. GraphX provides ETL, exploratory analysis, and iterative graph computation to enable users to interactively build and transform a graph data structure at scale. It comes with a highly flexible API and a selection of distributed Graph algorithms.

Spark Streaming is a real-time solution that uses Spark Core's fast scheduling capability to do streaming analytics. Spark Streaming ingests data in mini-batches and enables analytics on that data with the same application code that is written for batch analytics. This improves developer productivity because they can use the same code for batch processing and real-time streaming applications.

Spark MLlib is a library of algorithms to do ML on data at scale. Data scientists can use R or Python to train ML models on any Hadoop data source, save them using MLlib, and import them into a Java or Scala-based pipeline.

Key takeaways: Apache Spark

- Apache Spark performs processing in-memory, reduces the number of steps in a job, and reuses data across multiple parallel operations.
- Spark reuses data by using an in-memory cache to speed up ML algorithms.

|Slide number 23
|Instructor notes
|
|Student notes
Here are a few key points to summarize this section.

Apache Spark addresses some limitations of Apache Hadoop by performing processing in-memory, reducing the number of steps in a job, and reusing data across multiple parallel operations.

Spark reuses data by using an in-memory cache to speed up ML algorithms. Spark uses a collection of objects in cached memory known as DataFrames.

|Slide number 24
|Instructor notes
|
|Student notes
This section introduces the Amazon EMR service.

Amazon EMR characteristics

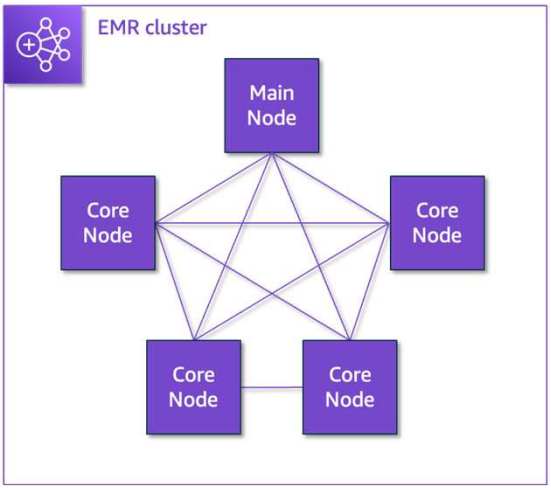
- Managed cluster platform
- Big data solution for petabyte-scale data processing, interactive analytics, and machine learning
- Processes data for analytics and BI workloads using big data frameworks
- Transform and move large amounts of data into and out of AWS data stores

|Slide number 25
|Instructor notes
|
|Student notes
Amazon EMR is a managed cluster platform. The service simplifies running big data frameworks, such as Apache Hadoop and Apache Spark, on AWS to process and analyze vast amounts of data. By using these frameworks and related open-source projects, such as Apache Hive and Apache Pig, you can process data for analytics and business intelligence (BI) workloads. Additionally, you can use Amazon EMR to transform and move large volumes of data. You can move this data in and out of other AWS data stores and databases, such as Amazon S3 and Amazon DynamoDB.

With Amazon EMR, you have the ability to launch clusters in minutes, deploy multiple clusters, and resize clusters that are already running. By using these clusters, you can process your large volumes of data in a cost-effective manner. Amazon EMR relies on HDFS and Amazon S3 for storage. Therefore, you can store input and output data in Amazon S3 while using HDFS to store your intermediate results. Finally, Amazon EMR provides the ability to perform automated installations of common big data projects such as Hive, Pig, and Spark, and run applications such as Hadoop, Spark, and Presto.

Clusters and nodes

- The central component of Amazon EMR is the *cluster*.
- Each instance in the cluster is a *node*.
- The role that each node serves is the *node type*.
- Amazon EMR uses three node types: main, core, and task.



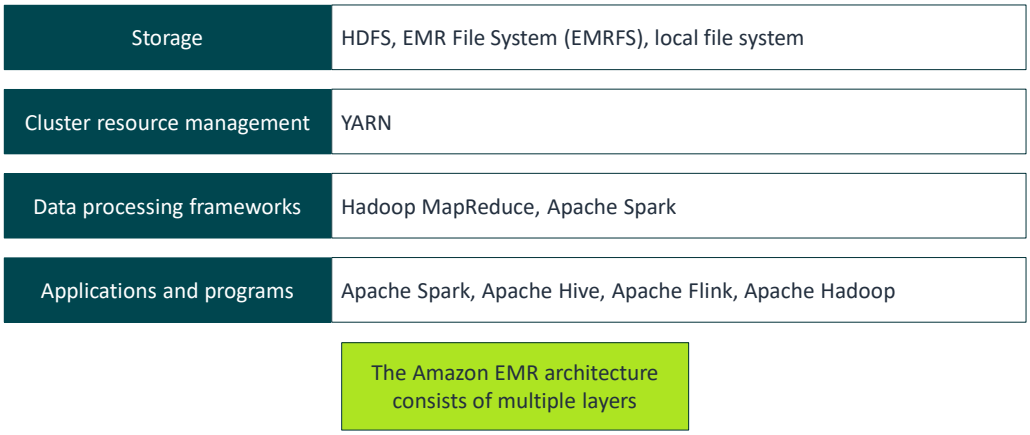
|Slide number 26
|Instructor notes
|
|Student notes
The central component of Amazon EMR is the *cluster*, which is a collection of Amazon Elastic Compute Cloud (Amazon EC2) instances. These EC2 instances are referred to as *nodes*, and each node has a role—known as *node type*—within the cluster. Each node type can have different software components, which further delineates its role in the cluster. Amazon EMR uses three node types: main, core, and task.

The *main node* is responsible for managing the cluster by running software components that coordinate data distribution and tasks to other nodes for processing. Every cluster has a main node, and you can create single-node clusters that have only a main node.

Core nodes are responsible for running tasks and storing data in your cluster's HDFS. Multi-node clusters must have at least one core node.

Task nodes, which are optional, only run tasks. Task nodes don’t store data in HDFS, and you can add them at cluster launch or add them to an already running cluster to increase processing capacity. Additionally, you can terminate individual task nodes to reduce costs when they are no longer needed.

Amazon EMR service architecture



|Slide number 27
|Instructor notes
|
|Student notes
Several layers make up the Amazon EMR service architecture. Each layer provides specific capabilities and functions to the cluster.

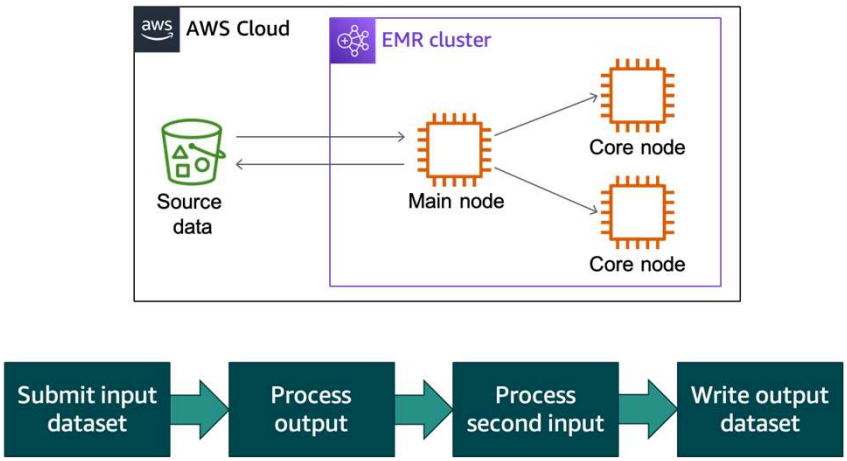
The *storage layer* consists of the different file systems that are used within your cluster, such as HDFS and EMRFS. This layer also includes the local file system of any locally connected disk. Each EC2 instance within the cluster has preconfigured block storage, known as an *instance store*, which persists for the duration of the instance's existence.

The *cluster resource management layer* is responsible for managing your cluster resources and scheduling data processing jobs. YARN is the default management component that Amazon EMR uses. Note that this doesn't mean that Amazon EMR clusters are restricted to YARN. Many frameworks and applications that are offered in Amazon EMR don't use YARN as a resource manager.

In the *data processing frameworks layer*, the main processing frameworks are Hadoop MapReduce and Spark.

The *applications and programs layer* is where the applications that were discussed earlier in this module provide capabilities such as using higher level languages to create processing workloads, using ML algorithms, making stream processing applications, and building data warehouses. In addition, Amazon EMR also supports open-source projects that have their own cluster management functionality instead of using YARN.

Processing data in Amazon EMR



|Slide number 28
|Instructor notes
|
|Student notes
When you launch your cluster, you choose the frameworks and applications to install for your data processing needs. To process data in your Amazon EMR cluster, you can submit jobs or queries directly to installed applications, or you can run *steps* in the cluster. You can submit jobs and interact directly with the software that is installed in your Amazon EMR cluster. To do this, you typically connect to the main node over a secure connection and access the interfaces and tools that are available for the software that runs directly on your cluster.

You can submit one or more ordered steps to an Amazon EMR cluster. Each step is a unit of work that contains instructions to manipulate data for processing by software that is installed on the cluster.

The slide illustrates an example process that uses four steps:
1. Submit an input dataset for processing.
2. Process the output of the first step by using a Pig program.
3. Process a second input dataset by using a Hive program.
4. Write an output dataset.

Generally, when you process data in Amazon EMR, the input is data that is stored as files in your chosen underlying file system, such as Amazon S3 or HDFS. This data passes from one step to the next in the processing sequence. The final step writes the output data to a specified location, such as an S3 bucket.

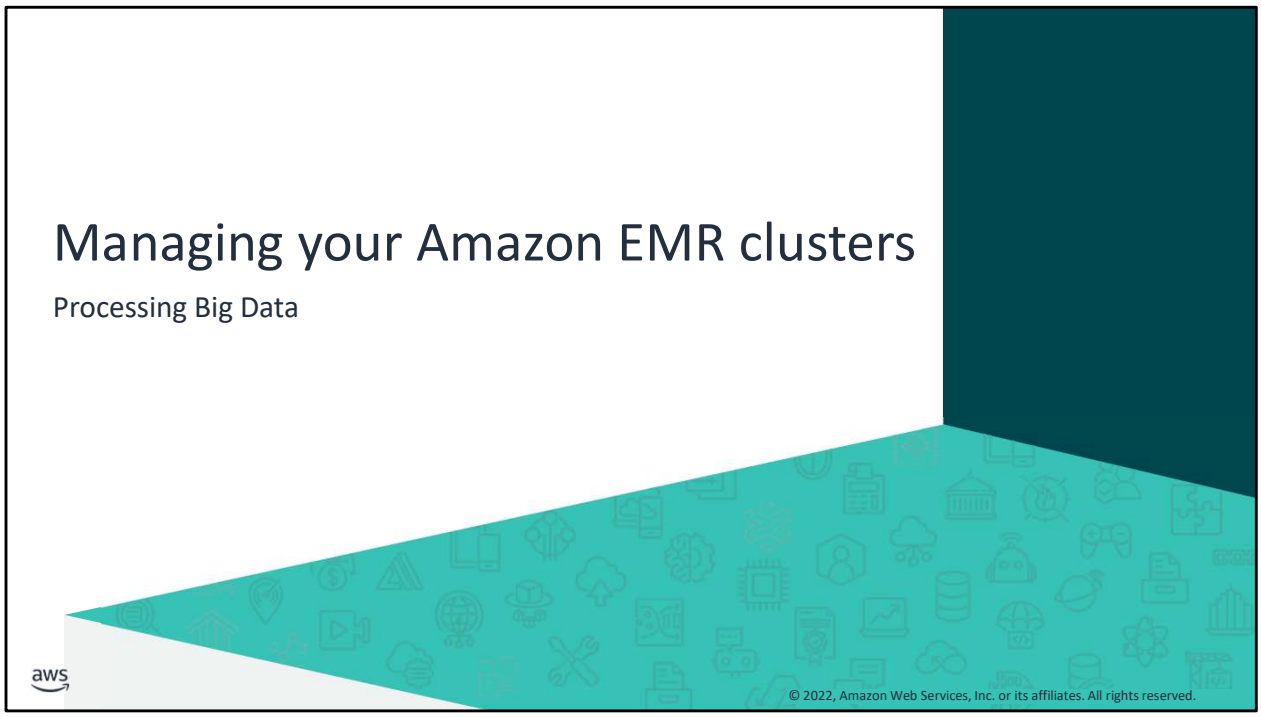
Key takeaways: Amazon EMR

- You can use Amazon EMR to perform automated installations of common big data projects.
- The Amazon EMR service architecture consists of four distinct layers:
  - Storage layer
  - Cluster resource management layer
  - Data processing frameworks layer
  - Applications and programs layer

|Slide number 29
|Instructor notes
|
|Student notes
Here are a few key points to summarize this section.

You can use Amazon EMR to perform automated installations of clusters for petabyte-scale data processing, interactive analytics, and ML projects.

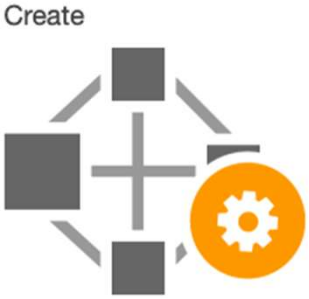
The Amazon EMR service architecture consists of the storage layer, cluster resource management layer, data processing frameworks layer, and applications and programs layer.



|Slide number 30
|Instructor notes
|
|Student notes
This section introduces concepts that are related to managing your Amazon EMR clusters.

Launching and configuring Amazon EMR clusters

- Three methods are available to launch EMR clusters:
  - Interactive mode
  - Command line mode
  - API mode



|Slide number 31
|Instructor notes
|
|Student notes
You can launch Amazon EMR clusters in three ways: interactive mode, command line mode, or API mode. Let's take a closer look at each method.

Interactive mode: You can create an EMR cluster in the AWS Management Console in a few steps. Within the Amazon EMR console, you can manually launch and configure a cluster by choosing **Create cluster** and completing the required fields. This configuration is referred to as the *Quick Options* method because it uses default values for many of the configuration options.

Command line mode: You can also launch and configure an EMR cluster by using the AWS Command Line Interface (AWS CLI). The AWS CLI provides Amazon EMR specific commands to launch, configure, and manage your clusters.

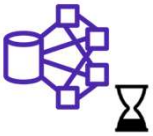
API mode: Finally, you can launch and configure a cluster by using the Amazon EMR API. Amazon EMR has a full range of supported API actions, such as *RunJobFlow*, which creates and runs a new cluster.

Cluster characteristics

Amazon EMR clusters are characterized in one of two ways:

Long-running clusters

- Persistent clusters
- Interactive jobs submission
- Persistent data (until shutdown)
- Large dataset processing



Transient clusters

- Shut down after data is processed and stored
- Typically read code and data from Amazon S3 at startup
- Don't persist HDFS data after termination

aws

|Slide number 32
|Instructor notes
|
|Student notes

When an Amazon EMR cluster is used for long-running analytics processes, it's referred to as a *long-running* or *persistent* cluster. When a cluster exists just long enough to complete a processing job or jobs, it's referred to as a *transient cluster*.

Long-running clusters

By default, clusters that you create using the console or AWS CLI are long running. Long-running clusters continue to run, accept work, and accrue charges until you shut them down.

A long-running cluster is effective in the following situations:

- When you need to interactively or automatically query data
- When you need to interact with big data applications that are hosted on the cluster on an ongoing basis
- When you periodically process a dataset so large or so frequently that it's inefficient to launch new clusters and load data each time

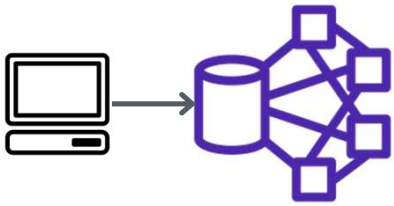
You can also set termination protection on a long-running cluster to avoid shutting down EC2 instances by accident or error.

Transient clusters
When you configure termination after step processing, the cluster starts, runs bootstrap actions, and then runs the steps that you specify. As soon as the last step completes, Amazon EMR terminates the cluster's EC2 instances. Clusters that you launch using the Amazon EMR API have step processing enabled by default.

Termination after step processing is effective for clusters that perform a periodic processing task, such as a daily data processing run. Step execution also helps you ensure that you are billed only for the time that is required to process your data.

Connecting to your cluster

- External connections are made to the main node Amazon EC2 instance.
  - The main node exposes the public DNS name for connections.
  - Amazon EMR creates security group rules for the main node.
  - Amazon EMR creates separate security group rules for the core and task nodes.
- Clusters must be running for connections to be made.



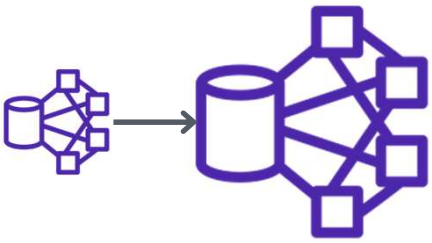
|Slide number 33
|Instructor notes
|
|Student notes
In an EMR cluster, the main node is an EC2 instance that coordinates the EC2 instances that are running as task and core nodes. The main node exposes a public DNS name, which you can use to connect to it. By default, Amazon EMR creates security group rules for the main node—and for core and task nodes—that determine how you access the nodes.

You can connect to the main node only while the cluster is running. When the cluster terminates, the EC2 instance that acts as the main node is terminated and is no longer available. To connect to the main node, you must also authenticate to the cluster. You can either use Kerberos for authentication or specify an Amazon EC2 key-pair private key when you launch the cluster. When you launch a cluster from the console, the EC2 key-pair private key is specified in the **Security and access** section on the **Create cluster** page.

By default, the ElasticMapReduce-main security group doesn't permit inbound SSH access. You might need to add an inbound rule that allows SSH access (TCP port 22) from the sources that need access.

Scaling your cluster resources

- Scaling is accomplished automatically or manually.
- Two options are available for automatic scaling:
  - Amazon EMR managed scaling
  - Custom automatic scaling policy



|Slide number 34
|Instructor notes
|
|Student notes
You can adjust the number of EC2 instances that are available to an Amazon EMR cluster automatically or manually in response to workloads that have varying demands. To use automatic scaling, you have two options: You can enable EMR managed scaling or create a custom automatic scaling policy.

When you scale EMR cluster resources, there are a few considerations. Amazon EMR clusters always consist of one or three main nodes. You can't scale the number of main nodes after your cluster has been configured, although you can scale core and task nodes within your cluster. Another consideration is that you can't reconfigure and resize an instance group at the same time. This means that if you initiate a resize while an instance group is reconfiguring, the resize can't occur until the reconfiguration has completed.

Key takeaways: Managing your Amazon EMR clusters

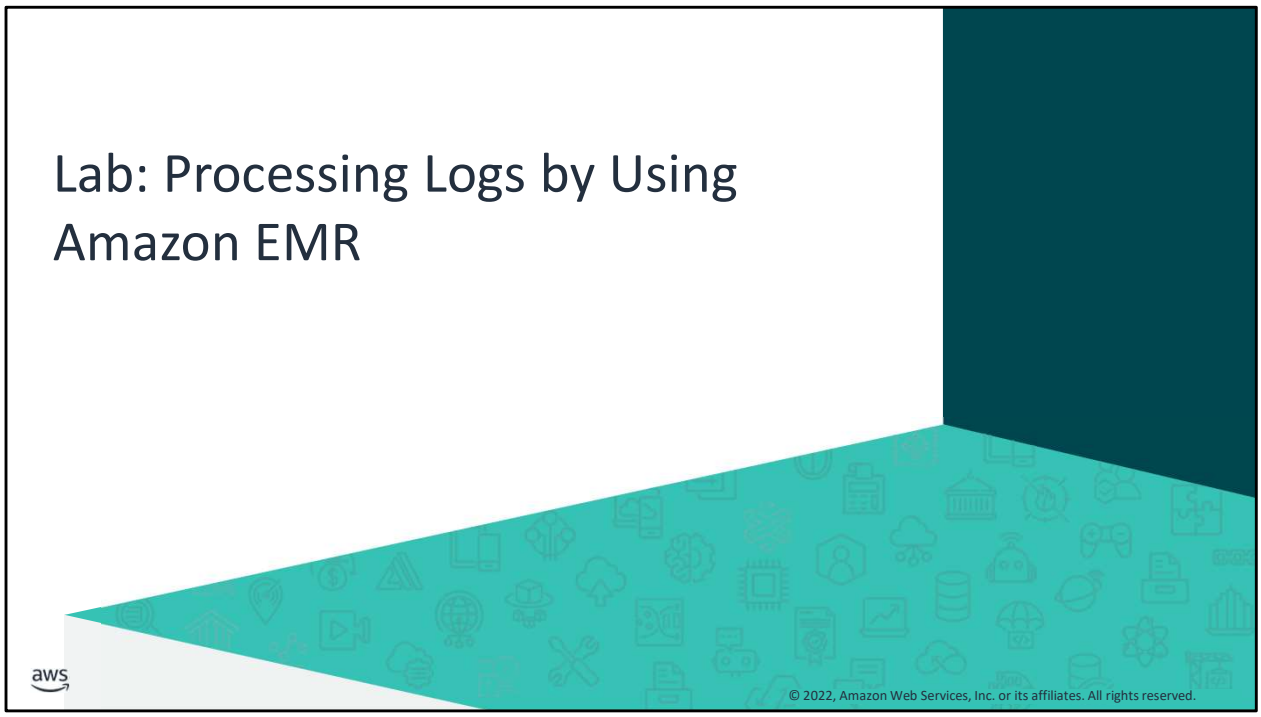
- Three methods are available to launch EMR clusters: interactive, command line, and API.
- EMR clusters are characterized as either long running or transient, based on their usage.
- External connections to EMR clusters can only be made through the main node.

|Slide number 35
|Instructor notes
|
|Student notes
Here are a few key points to summarize this section.

Three methods are available to launch EMR clusters: interactive mode, command line mode, and API mode.

Amazon EMR clusters are classified as long-running by default, but they can be considered as transient if they are set to shut down after data is finished processing.

All external connections to EMR clusters must be made through the main node. A direct method of communicating with core nodes isn't available—other than through the main node.



|Slide number 36
|Instructor notes
|
|Student notes
You will now complete a lab. The next slide summarizes what you will do in the lab, and you will find the detailed instructions in the lab environment.

Lab introduction: Processing Logs by Using Amazon EMR

- In this lab, you will use Amazon EMR to process logs that are stored in an S3 bucket.
- To process the data in Amazon EMR, you will use Apache Hive.
- Open your lab environment to start the lab and find additional details about the tasks that you will perform during this lab.

|Slide number 37
|Instructor notes
|Give directions to find the lab guide information in the instructor notes section.
|
|Student notes
Access the lab environment through your online course to get additional details and complete the lab.

Debrief: Processing Logs by Using Amazon EMR

- In Hive, what is the difference between an external table and a table that isn't defined as external?
- What happens when you run a SELECT query in Hive?
- Are AWS Glue and Athena comparable to Hive? How are they similar or different?

|Slide number 38
|Instructor notes
|Q1 - **Example strong response:** With an external table, the table data stays in Amazon S3 and isn't copied into HDFS. However, with a table that isn't external, the data is copied into HDFS. If you drop an external table, the table data itself will remain in Amazon S3.
|Q2 - **Example strong response:** A SELECT query is translated into a distributed MapReduce application. The main node distributes the application to the core nodes.
|Q3 - **Example strong response:** An external table that is defined in Hive can define the structure of data that is stored in Amazon S3. An AWS Glue table can also define the structure of data that is stored in S3. You could use Athena to query data in a Hive table. You can also configure Hive to use the AWS Glue Data Catalog as its metastore. With both Athena and Hive, you can run SQL queries on data in S3. However, with Hive, you could also query data that is stored in the EMR cluster itself. With Athena, you cannot query data in an EMR cluster. With Hive, the processing power that is available depends on the size of the EMR cluster. With Athena, you don't need to manage a cluster.
|
|Student notes
Your instructor might review these questions with you, or you might review them on your own. Use this opportunity to extend your thinking about the tasks that you performed during the lab.

|Slide number 39
|Instructor notes
|
|Student notes
This section introduces concepts related to the Apache Hudi framework.

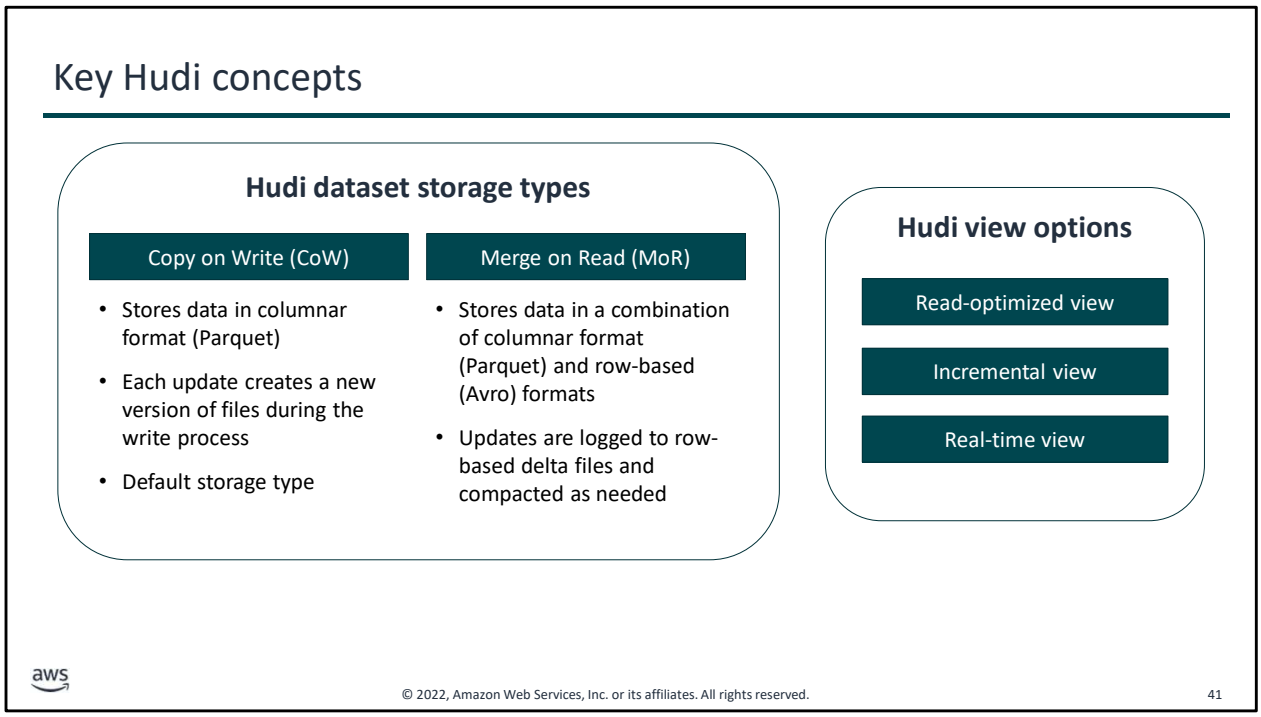
Apache Hudi characteristics

- Is an open-source data management framework
- Provides record-level insert, update, upsert, and delete capabilities
- Integrates with Apache Spark, Apache Hive, and Presto
- Hudi DeltaStreamer utility provides the ability to create or update Hudi datasets
- Organizes a dataset into a partitioned directory structure under a base path, similar to a Hive table

|Slide number 40
|Instructor notes
|
|Student notes
Apache Hudi is an open-source data management framework. Hudi is 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 don't already exist or to update them if they do. Hudi allows data to be ingested and updated in near real time by managing how data is organized in Amazon S3. Hudi maintains metadata of the actions that are performed on the dataset to help ensure that the actions are both atomic and consistent.

Hudi is integrated with Apache Spark, Apache Hive, and Presto. Amazon EMR installs Hudi components by default when Spark, Hive, Presto, or Flink are installed. You can use Spark or the Hudi DeltaStreamer utility to create or update Hudi datasets. You can use Hive, Spark, Presto, or Flink to query a Hudi dataset interactively. You can also build data processing pipelines using the *incremental pull*, which allows you to pull only the data that changed between two actions.

These features make Hudi suitable for working with streaming data from sensors and other Internet of Things (IoT) devices that require specific data insertion and update events. Additionally, Hudi is suitable for maintaining compliance with data privacy regulations in applications where users might choose to be forgotten or to modify their consent for how their data can be used.



|Slide number 41
|Instructor notes
|
|Student notes
When you create your Hudi dataset, you must specify whether it will be Copy on Write (CoW) or Merge on Read (MoR).

CoW stores your data in a columnar format—specifically, the Apache Parquet format. Each update to the data creates a new version of files during the write process. CoW is the default type of storage for Hudi, and it is well-suited for read-heavy workloads where data changes infrequently.

MoR stores your data using a combination of columnar and row-based formats—Apache Parquet and Apache Avro, respectively. Updates are logged to row-based delta files and are compacted as needed to create new versions of the columnar files. MoR is well-suited for workloads where writes and changes are frequent and reads are less frequent.

Hudi provides you with three logical views to access your data: read-optimized view, incremental view, and real-time view. The read-optimized view provides the latest committed dataset from CoW tables, as well as the latest compacted dataset from MoR tables. The incremental view provides a change stream between two actions from a CoW dataset that can be fed to downstream jobs and ETL workflows. Finally, the real-time view provides the latest committed data from a MoR table by merging the columnar and row-based files inline.

Key takeaways: Apache Hudi

- Apache Hudi provides the ability to ingest and update data in near real time.
- Hudi maintains metadata of the actions that are performed to ensure those actions are both atomic and consistent.

|Slide number 42
|Instructor notes
|
|Student notes
Here are a few key points to summarize this section.

Hudi provides the ability to ingest and update data in near real time by managing how data is organized in Amazon S3.

Hudi maintains metadata of the actions that are performed on the dataset to help ensure that the actions are both atomic and consistent.

|Slide number 43
|Instructor notes
|
|Student notes
You will now complete a lab. The next slide summarizes what you will do in the lab, and you will find the detailed instructions in the lab environment.

Lab introduction: Updating Dynamic Data in Place

- In this lab, you will use Amazon S3, Amazon Athena, AWS Glue, and Apache Hudi to address the challenge of accommodating a dynamically changing schema.
- You will use these services to facilitate efficient in-place data updates and run queries to get data in near real time.
- Open your lab environment to start the lab and find additional details about the tasks that you will perform during this lab.

|Slide number 44
|Instructor notes
|
|Student notes
Access the lab environment through your online course to get additional details and complete the lab.

Debrief: Updating Dynamic Data in Place

- How did you capture streaming data and store it in Amazon S3 in this lab?
- What is the purpose of the schema evolution capability in Apache Hudi?
- After you changed the schema of the data that Kinesis was publishing, how did you update the Athena query?

|Slide number 45
|Instructor notes
|Q1 - **Example strong response:** The lab used the Amazon Kinesis Data Generator (KDG) to emulate an IoT device and publish data to a Kinesis data stream. An AWS Glue job contained a Python script to capture the streaming data and insert it into an S3 bucket by using an Apache Hudi Connector.
|Q2 - **Example strong response:** Schema evolution enables a user to modify the table schema so that they can adapt to changes in data structure. The capability enables the AWS Glue job to handle records that don't all have the same data structure—for example, some records contain an additional column of data. **Note:** The lab does not explicitly point it out, but the feature is similar to the ALTER TABLE feature that is available in standard relational databases. However, the schema evolution feature is unique because it can be used with data stored in Amazon S3.
|Q3 - **Example strong response:** We added "new_column" to the SELECT statement in step 14. This attribute was not included in the query in step 13. Of course, we could have run "SELECT * FROM hudi_demo_table" in both steps instead, and that would simply return all the columns that are defined in the table.
|
|Student notes
Your instructor might review these questions with you, or you might review them on your own. Use this opportunity to extend your thinking about the tasks that you performed during the lab.

|Slide number 46
|Instructor notes
|
|Student notes
This section summarizes what you have learned and brings the module to a close.

Module summary

This module prepared you to do the following:

- Compare and select the big data processing framework that best supports your workloads.
- Explain the principles of Apache Hadoop and Amazon EMR, and how they support data processing in AWS.
- Launch, configure, and manage an Amazon EMR cluster to support big data processing.

|Slide number 47
|Instructor notes
|This is a good opportunity to use an online group or discussion board to ask students to reflect on what they have learned. You might ask the students to recall a point from the module that aligns to one of the listed objectives. This provides a good segue to the knowledge check and sample exam question.
|
|Student notes
This module focused on the processing frameworks and AWS services that support big data processing. You learned how to compare and select the big data processing framework that best supports your workloads. You learned about Apache Hadoop and Amazon EMR, and how they support data processing in AWS. Finally, you learned how to launch, configure, and manage an Amazon EMR cluster to support big data processing.

Module knowledge check

- The knowledge check is delivered online within your course.
- The knowledge check includes 10 questions based on material presented on the slides and in the slide notes.
- You can retake the knowledge check as many times as you like.

|Slide number 48
|Instructor notes
|
|Student notes
Use your online course to access the knowledge check for this module.

Sample exam question

A data engineer has deployed an Amazon EMR cluster to support their ML workload. Additional cluster configuration is needed, but numerous attempts to use SSH to connect to the active EC2 instance, which is acting as the main node of the cluster, have failed.

What could cause this connectivity issue?

Identify the key words and phrases before continuing.

The following are the key words and phrases:

- **Amazon EMR cluster**
- **SSH**
- **Connect to the active EC2 instance**
- **Main node of the cluster**

aws

|Slide number 49

|Instructor notes: The key words section is animated to be revealed on click.

|

|Student notes

The question notes that the engineer isn't able to use SSH to connect to an active EC2 instance that is acting as the main node of their Amazon EMR cluster.

Sample exam question: Response choices

A data engineer has deployed an **Amazon EMR cluster** to support their ML workload. Additional cluster configuration is needed, but numerous attempts to use **SSH** to **connect to the active EC2 instance**, which is acting as the **main node of the cluster**, have failed.

What could cause this connectivity issue?

Choice	Response
A	Connections to an EMR cluster must be made directly to the desired core node.
B	Amazon EMR does not support SSH connections to clusters.
C	The ElasticMapReduce-main security group needs an inbound rule that allows HTTPS access.
D	The ElasticMapReduce-main security group needs an inbound rule that allows SSH access.

|Slide number 50
|Instructor notes
|
|Student notes

Use the key words that you identified on the previous slide, and review each of the possible responses to determine which one best addresses the question.

Sample exam question: Answer

The correct answer is D.

Choice	Response
D	The ElasticMapReduce-main security group needs an inbound rule that allows SSH access.

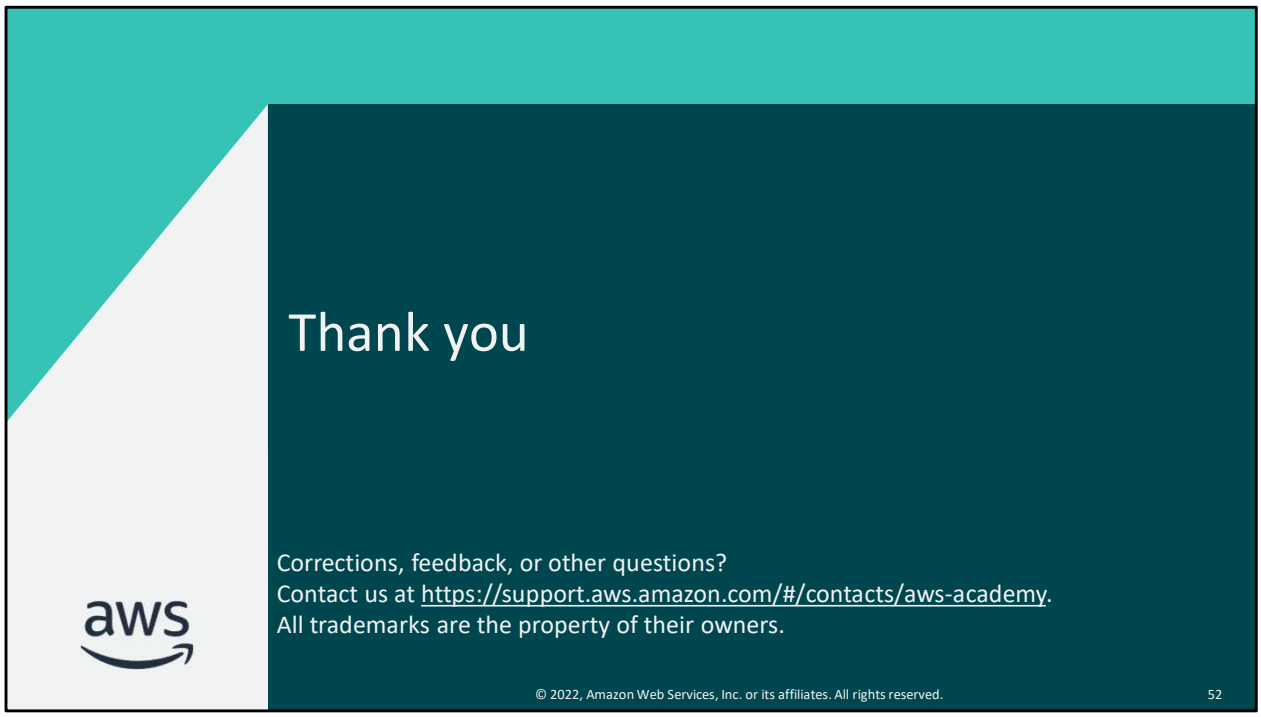
|Slide number 51
|Instructor notes
|
|Student notes

Choice A (Connections to an EMR cluster must be made directly to the desired core node) is incorrect. Connections to EMR clusters are made to the main node, not cluster nodes.

Choice B (Amazon EMR does not support SSH connections to clusters) is also incorrect. Connections to EMR clusters are made using the SSH protocol to the EC2 instance that acts as the main node.

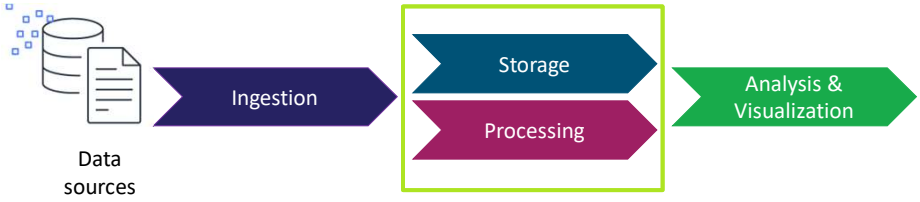
Choice C (The ElasticMapReduce-main security group needs an inbound rule that allows HTTPS access) is incorrect. Amazon EMR does not use HTTPS connections for management purposes.

Choice D (The ElasticMapReduce-main security group needs an inbound rule that allows SSH access) is the correct choice. By default, the ElasticMapReduce-main security group does not contain an inbound rule that allows SSH access, which will restrict inbound SSH connections.



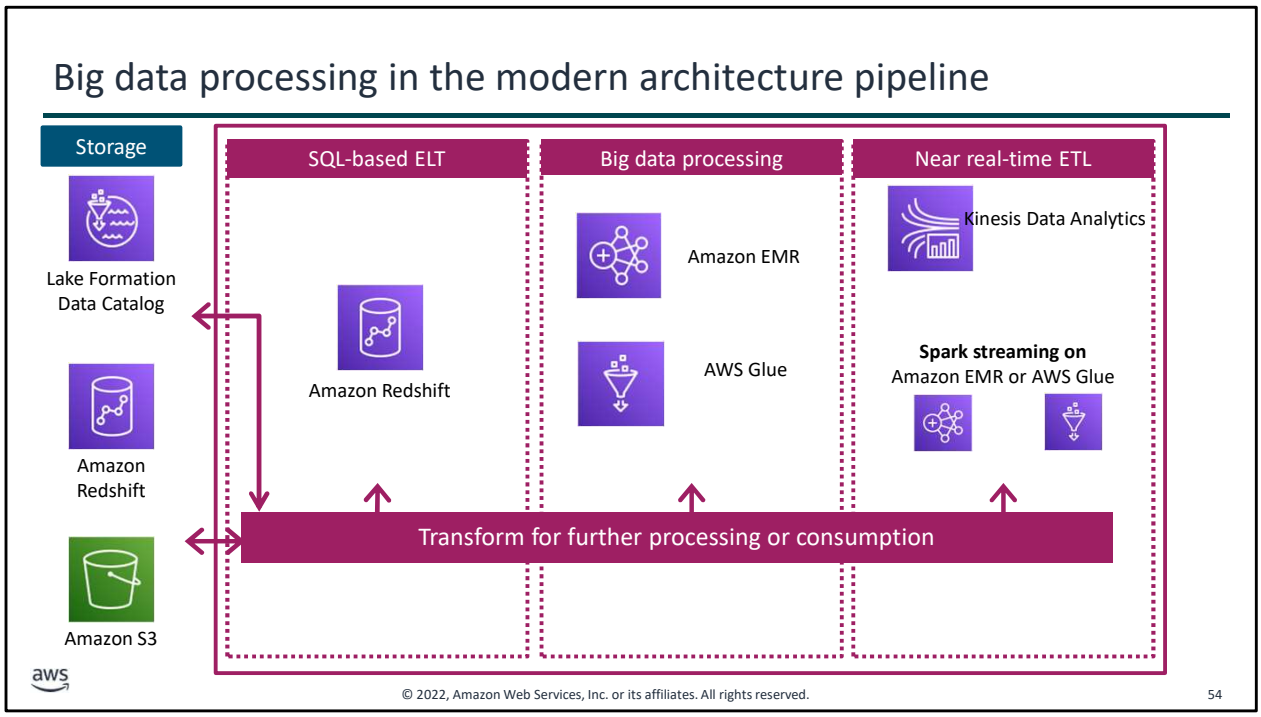
|Slide number 52
|Instructor notes
|
|Student notes
That concludes this module. The Content Resources page of your course includes links to additional resources that are related to this module.

Big data processing and the iterative data pipeline



|Slide number 53
|Instructor notes
|
|Student notes
Up to this point in the course, you have learned about the roles that ingestion and storage play in the pipeline. You also learned how closely related they are to the processing phase of the iterative data pipeline. Data frequently flows between the processing and storage elements. Raw data is ingested, stored, transformed, and processed. That data often returns to storage in a new format, and that formatted data is used for analysis and visualization.

In this module, you will learn about some of the common frameworks for processing big data and the services that are used to process data for your analytics and machine learning (ML) pipelines.

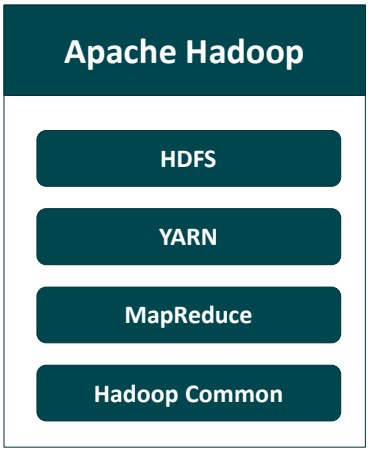


|Slide number 54
|Instructor notes
|
|Student notes
In the Design Principles and Patterns for Data Pipelines module, you learned that components in the data processing layer are responsible for transforming data into a consumable state. The processing layer provides purpose-built components that enable a variety of data types, velocities, and transformations. Each component can read and write data to both Amazon S3 and Amazon Redshift in the storage layer, and all can scale to high data volumes.

This architecture supports multiple ETL or ELT pipelines that perform iterative processing of data for different types of preparation and consumption. Each pipeline reads data from the storage layer, processes it using temporary storage as needed, and then writes it to the appropriate location within the storage layer. This module focuses on the big data processing aspect of this architecture.

Apache Hadoop characteristics

- Is an open-source, distributed processing framework
- Enables distributed storage and processing for large amounts of data
- Maps tasks to nodes within clusters of servers
- Hadoop components: Hadoop Distributed File System (HDFS), YARN, MapReduce, Hadoop Common
- Clusters consist of main nodes and worker nodes



aws

|Slide number 55
|Instructor notes
|
|Student notes
Hadoop is an open-source framework that uses a distributed processing architecture. Hadoop maps tasks to a cluster of commodity servers for processing. Hadoop consists of four main components: Hadoop Distributed File System (HDFS); Yet Another Resource Negotiator (YARN); MapReduce; and Hadoop Common. Hadoop clusters consist of main nodes and worker nodes. In this format, main nodes are responsible for orchestrating jobs, while worker nodes are responsible for processing those jobs.

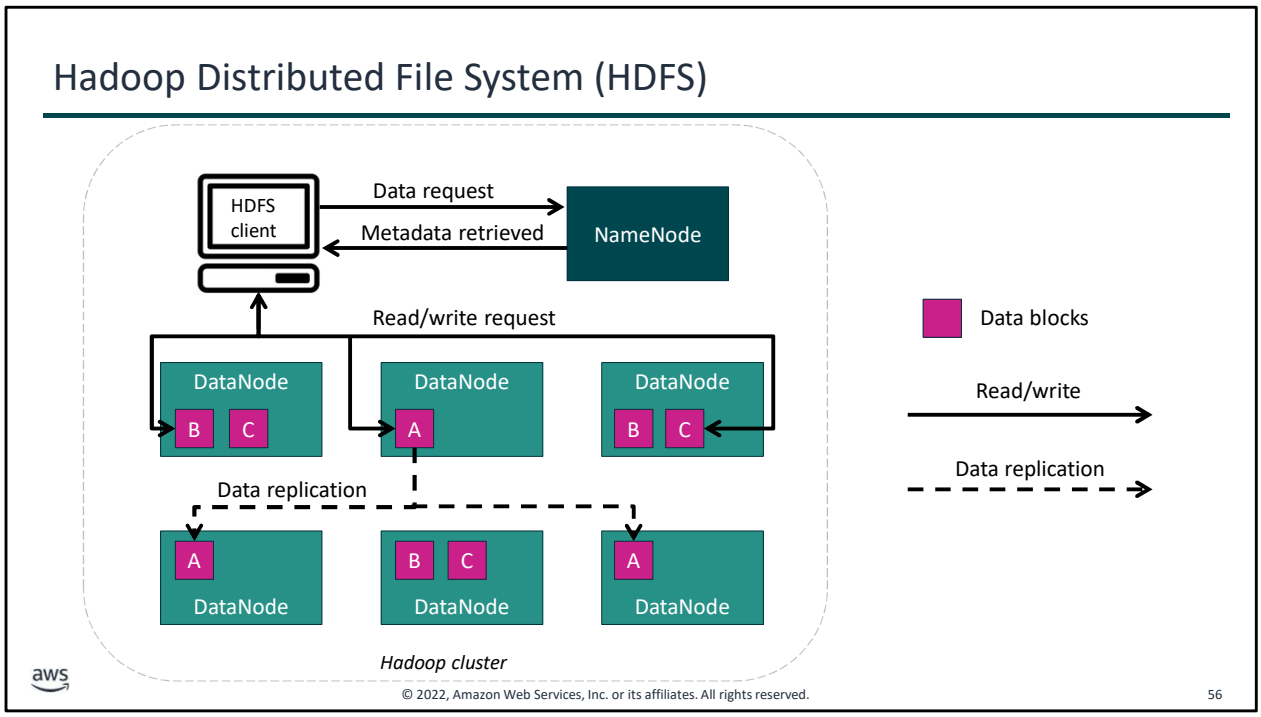
Benefits of the Hadoop framework
With Hadoop, you can store as much data as you like. You aren't required to preprocess data prior to storing it; therefore, the framework provides increased flexibility. Hadoop also has a high degree of fault tolerance, thanks to node failover. If a node fails within a cluster, its tasks are redistributed throughout the other nodes within the cluster. Data loss is likewise prevented by having multiple copies of the same data stored throughout the cluster.

The Hadoop infrastructure is scalable—as data processing and computing requirements

grow, you can easily add nodes to support that growth in volume and velocity. Because it is an open-source framework, Hadoop is free and capable of running on inexpensive hardware. Finally, you can use Hadoop to process structured, unstructured, and semistructured data. You can transform the data into a variety of other formats to integrate with your existing datasets, and you can store the data with or without a schema.

Challenges of the Hadoop framework

The framework does have a few challenges. Because of its open-source nature, stability issues can arise as the framework is updated. You can avoid this challenge by running only the latest stable version. Hadoop also makes heavy use of the Java programming language—a commonly targeted language that can leave the framework open to vulnerabilities. Lastly, security concerns are inherent to Hadoop. Lack of authentication and encryption can lead to questions about the veracity of the data.



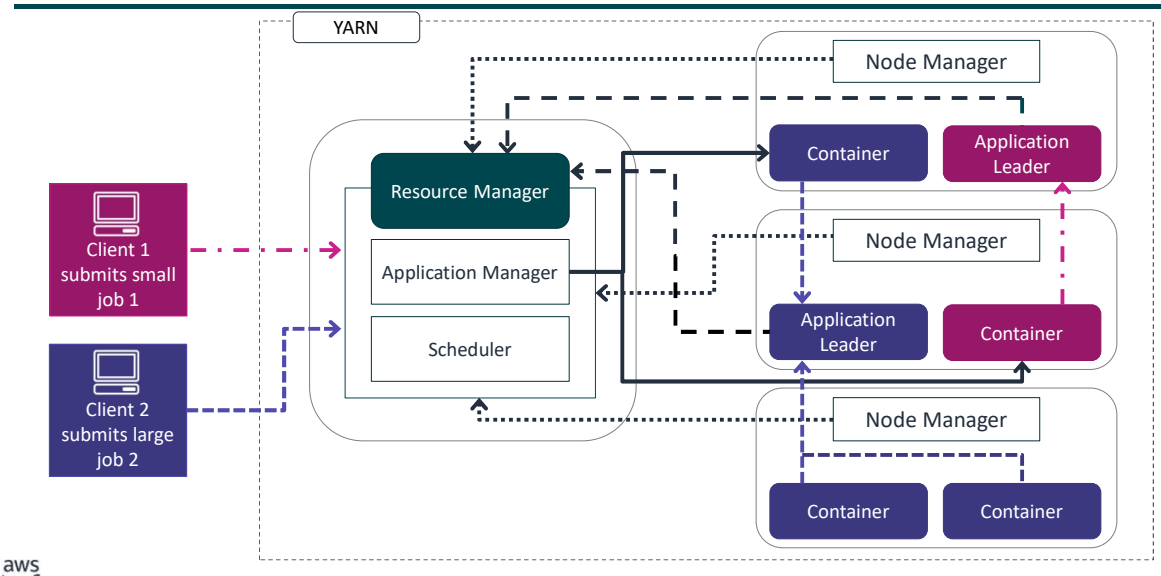
|Slide number 56
|Instructor notes
|
|Student notes
The Hadoop Distributed File System, or HDFS, is the distributed file system of the Hadoop framework. You can use HDFS to store huge amounts of data for further processing. HDFS and the Hadoop cluster use a hierarchical node architecture, which consists of a single NameNode that manages multiple DataNodes across clusters.

HDFS is typically used by applications that have large datasets, so it's designed to provide high aggregate data bandwidth. To support data velocity while storing data, HDFS splits the data into small blocks—called data blocks—and stores those blocks across several nodes of the cluster. In the example on this slide, a large data file has been split into three data blocks.

Each block resides in a different data node when possible. To avoid losing data if a cluster node fails, HDFS replicates each block several times across different nodes. This enables a high degree of fault tolerance within HDFS. The number of times that a block is replicated is called the *replication factor*, and it's a configurable setting. The cluster's NameNode catalogs the metadata about each file that is stored in the cluster. The cluster DataNodes

store the data blocks for each file that has been saved to HDFS.

Yet Another Resource Negotiator (YARN)



|Slide number 57
|Instructor notes
|
|Student notes

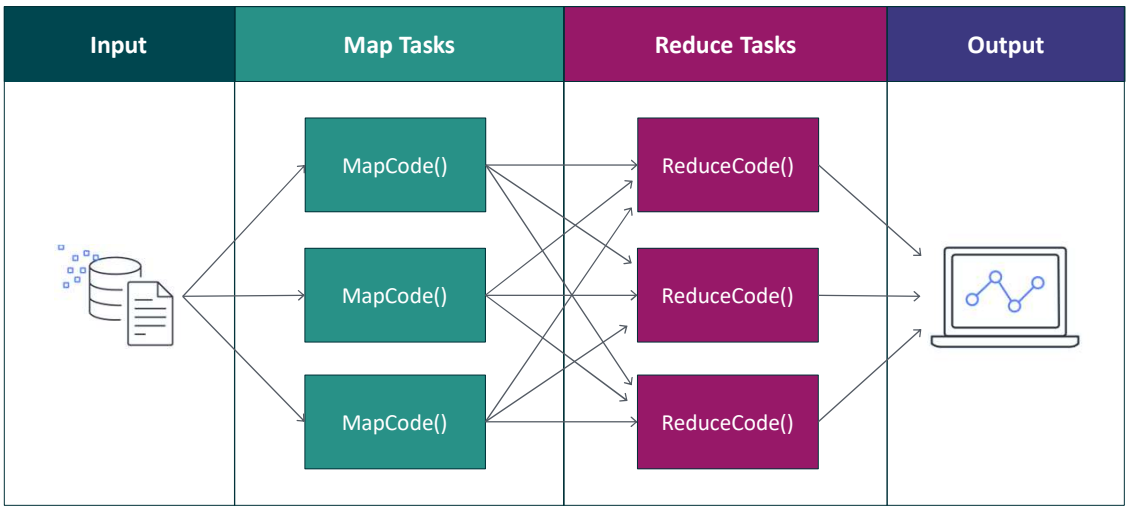
Yet Another Resource Negotiator—also called YARN—is a large-scale, distributed operating system that is used with Hadoop. YARN makes the data that is stored in HDFS accessible for different processing. YARN dynamically coordinates the use of available Hadoop resources to schedule and perform that processing.

YARN has a few primary components:

- **Resource Manager:** Controls the use of resources within the Hadoop cluster, and manages the containers that are launched on each cluster node. The Resource Manager has two main components:
    - **Scheduler:** Allocates resources to the running applications, based on the resource requirements of those applications.
    - **Application Manager:** Accepts job submissions, negotiates the first container to run the Application Leader, and provides the service to restart the Application Leader on failure.
- **Node Manager:** Controls the use of resources within a single Hadoop cluster node, and monitors the containers that are launched on that cluster node.

- **Application Leader:** Works with the Resource Manager and Node Manager to acquire cluster resources for processing tasks, before running and monitoring those tasks.
- **Containers:** Collections of cluster resources, such as memory and compute, that are allocated from a single cluster node to perform assigned processing activities.

Hadoop MapReduce

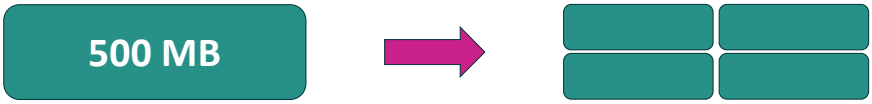


|Slide number 58
|Instructor notes
|
|Student notes
MapReduce is a framework for processing large datasets with a parallel, distributed algorithm on a cluster. Hadoop MapReduce simplifies writing parallel distributed applications by handling all of the logic, while you provide the Map and Reduce functions. The Map function maps data to sets of key-value pairs called *intermediate results*. The Reduce function combines the intermediate results, applies additional algorithms, and produces the final output. Multiple frameworks are available for MapReduce, such as Hive, which automatically generates Map and Reduce programs. The MapReduce framework is at the core of Hadoop and provides massive scalability across enormous numbers of Hadoop clusters. The framework is designed for fault tolerance, with each worker node periodically reporting its status to a leader node. The leader node can redistribute work from a cluster that doesn't respond positively.

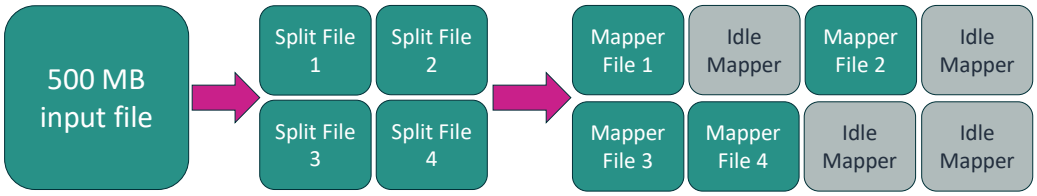
When running a big data job, the process begins as MapReduce splits the job into discrete tasks so that the tasks can run in parallel. Next, the mapper phase maps data to key-value pairs (for example, the number of occurrences of each word on a data block). As soon as the mapper phase is finished, the next step is to shuffle and sort the data. During this step,

for example, similar words are shuffled, sorted, and grouped together. The reduce phase counts the number of occurrences of words in the different groups and generates the output file.

Processing data with Hadoop MapReduce



A single file from Amazon S3 is split into four parallel HTTP requests



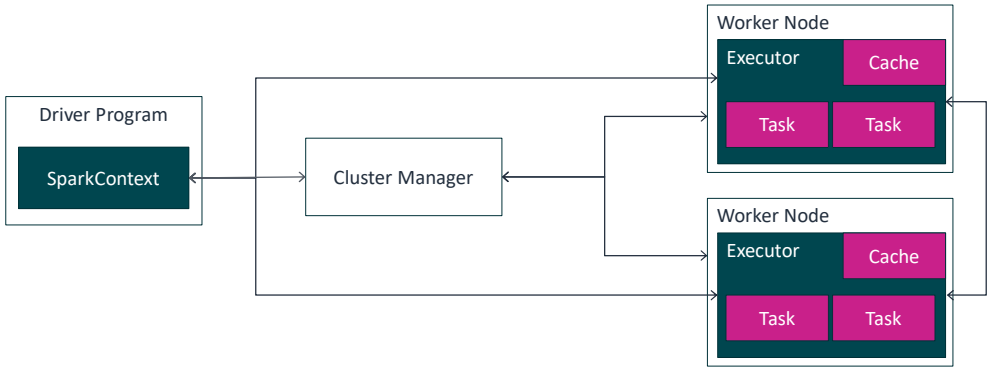
The Hadoop default split results in four files, using four of the eight available mappers

|Slide number 59
|Instructor notes
|
|Student notes

If you use HDFS to store your data, Hadoop automatically splits your data when it is stored in the HDFS cluster nodes. However, if you use Amazon Simple Storage Service (Amazon S3) to store your data, Hadoop splits the data by reading your files in multiple HTTP range requests whenever a processing job is started. The split size that Hadoop uses to read data from Amazon S3 varies depending on the Amazon EMR version that is being used. (Newer versions have larger split sizes.) The split size is generally the size of an HDFS block when operating on data that is stored in HDFS. Larger numbers provide less task granularity but also put less strain on the cluster NameNode.

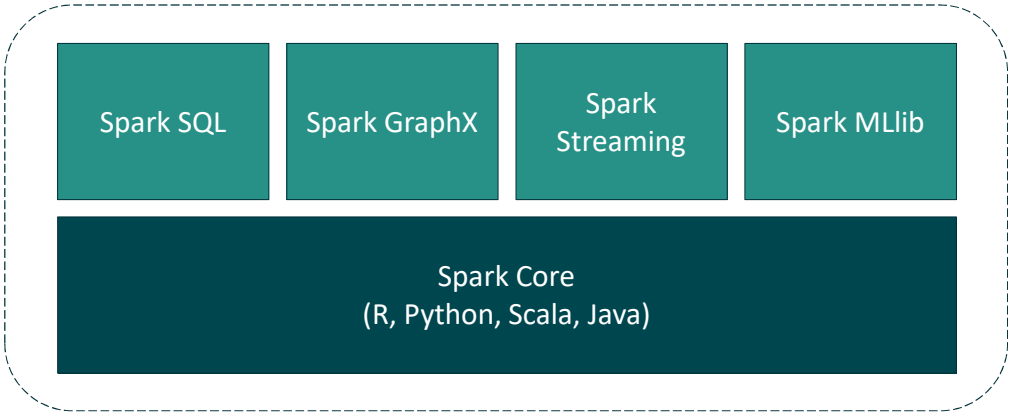
In the example on the slide, the default split size has been set to 134,217,728 bytes (128 MB). Eight mapper processes are available to process the data. Big data processing needs to maintain pace with the volume and velocity of data, so in this instance the 500 MB input file will be split into four smaller files. Each of those files will use an available mapper for processing in parallel.

Spark clusters



|Slide number 60
|Instructor notes
|
|Student notes
In Spark clusters, Spark applications run as independent sets of processes.
The *SparkContext* object that resides within the driver program coordinates these processes. Spark connects to a cluster manager, which acquires *executors* from nodes within the cluster. These executors are responsible for running the computations and storing the data in cache for your application. The SparkContext object communicates directly with the executors, sending them tasks to be processed.

Spark components



|Slide number 61
|Instructor notes
|
|Student notes
The components that make up the Spark framework are Spark Core, Spark SQL, Spark GraphX, Spark Streaming, and Spark MLlib. Let's take a closer look at each of these.

As the name implies, *Spark Core* is the foundation of the platform. It’s responsible for memory management, fault recovery, scheduling, distributing and monitoring jobs, and interacting with storage systems. Spark Core is exposed through APIs built for Java, Scala, Python, and R. These APIs hide the complexity of distributed processing behind simple, high-level operators.

Spark SQL is a distributed query engine that provides low-latency, interactive queries up to 100 times faster than MapReduce. Spark SQL includes a cost-based optimizer, columnar storage, and code generation for fast queries, while scaling to thousands of nodes.

Spark GraphX is a distributed graph processing framework that is built on top of Spark. GraphX provides ETL, exploratory analysis, and iterative graph computation to enable users to interactively build and transform a graph data structure at scale. It comes with a highly

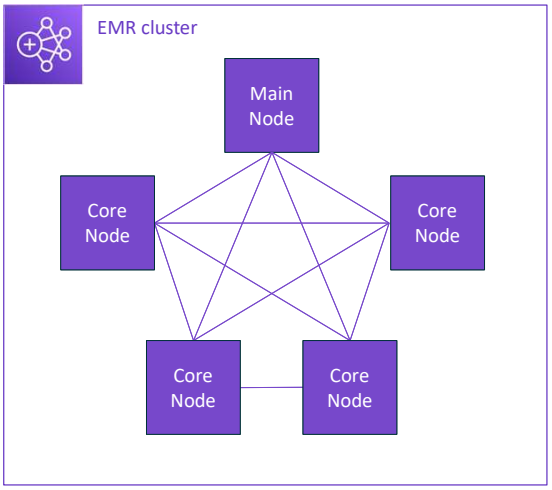
flexible API and a selection of distributed Graph algorithms.

Spark Streaming is a real-time solution that uses Spark Core's fast scheduling capability to do streaming analytics. Spark Streaming ingests data in mini-batches and enables analytics on that data with the same application code that is written for batch analytics. This improves developer productivity because they can use the same code for batch processing and real-time streaming applications.

Spark MLlib is a library of algorithms to do ML on data at scale. Data scientists can use R or Python to train ML models on any Hadoop data source, save them using MLlib, and import them into a Java or Scala-based pipeline.

Clusters and nodes

- The central component of Amazon EMR is the *cluster*.
- Each instance in the cluster is a *node*.
- The role that each node serves is the *node type*.
- Amazon EMR uses three node types: main, core, and task.



|Slide number 62
|Instructor notes
|
|Student notes
The central component of Amazon EMR is the *cluster*, which is a collection of Amazon Elastic Compute Cloud (Amazon EC2) instances. These EC2 instances are referred to as *nodes*, and each node has a role—known as *node type*—within the cluster. Each node type can have different software components, which further delineates its role in the cluster. Amazon EMR uses three node types: main, core, and task.

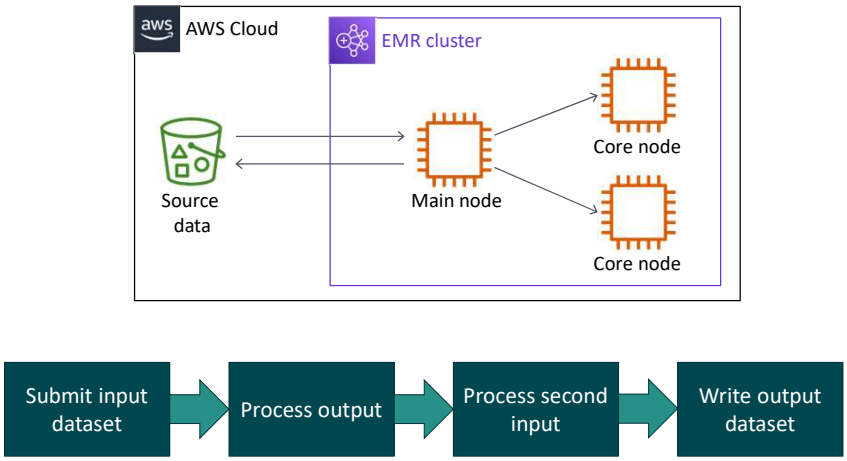
The *main node* is responsible for managing the cluster by running software components that coordinate data distribution and tasks to other nodes for processing. Every cluster has a main node, and you can create single-node clusters that have only a main node.

Core nodes are responsible for running tasks and storing data in your cluster's HDFS. Multi-node clusters must have at least one core node.

Task nodes, which are optional, only run tasks. Task nodes don’t store data in HDFS, and you can add them at cluster launch or add them to an already running cluster to increase processing capacity. Additionally, you can terminate individual task nodes to reduce costs

when they are no longer needed.

Processing data in Amazon EMR



|Slide number 63
|Instructor notes
|
|Student notes
When you launch your cluster, you choose the frameworks and applications to install for your data processing needs. To process data in your Amazon EMR cluster, you can submit jobs or queries directly to installed applications, or you can run *steps* in the cluster. You can submit jobs and interact directly with the software that is installed in your Amazon EMR cluster. To do this, you typically connect to the main node over a secure connection and access the interfaces and tools that are available for the software that runs directly on your cluster.

You can submit one or more ordered steps to an Amazon EMR cluster. Each step is a unit of work that contains instructions to manipulate data for processing by software that is installed on the cluster.

The slide illustrates an example process that uses four steps:
1. Submit an input dataset for processing.
2. Process the output of the first step by using a Pig program.
3. Process a second input dataset by using a Hive program.

4. Write an output dataset.

Generally, when you process data in Amazon EMR, the input is data that is stored as files in your chosen underlying file system, such as Amazon S3 or HDFS. This data passes from one step to the next in the processing sequence. The final step writes the output data to a specified location, such as an S3 bucket.