
UTEC Posgrado

Big Data Concepts

Juan pablo hernandez palacios

Big Data - Conceptos

- **Volume**: Volume refers to the sheer amount of data generated and collected. Big data is characterized by large volumes of data that exceed the capabilities of traditional data processing systems. This includes data from various sources such as social media, sensors, transaction records, and more.

- **Velocity**: Velocity represents the speed at which data is generated and processed. With the advent of real-time data streams, big data analytics focuses on analyzing data as it is generated to gain timely insights and make quick decisions. Velocity is crucial for applications that require real-time or near real-time processing.

Big Data - Conceptos

- **Variety**: Variety refers to the diverse types and formats of data that are encountered in big data. This includes structured data (e.g., relational databases), semi-structured data (e.g., XML, JSON), and unstructured data (e.g., text, images, videos). Big data platforms are designed to handle this variety of data types efficiently.

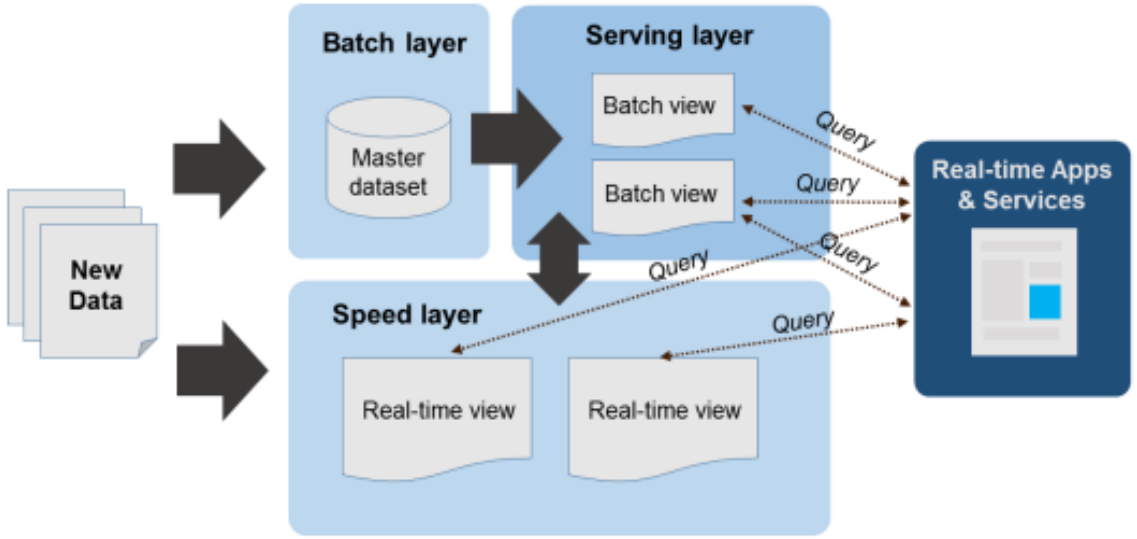
- **Veracity**: Veracity relates to the quality, reliability, and accuracy of the data. Big data often includes data from multiple sources, and it may contain inconsistencies, errors, or noise. Managing and validating the veracity of data is essential to ensure the accuracy of insights and decisions derived from big data analytics.

Big Data - Conceptos

- **Value**: Value represents the ultimate goal of big data analytics. Extracting meaningful insights and value from the data is the key objective. This involves employing various techniques such as data mining, machine learning, and predictive analytics to discover patterns, trends, correlations, and actionable insights that can drive informed decision-making and create business value.

Big Data – Arquitectura Lambda

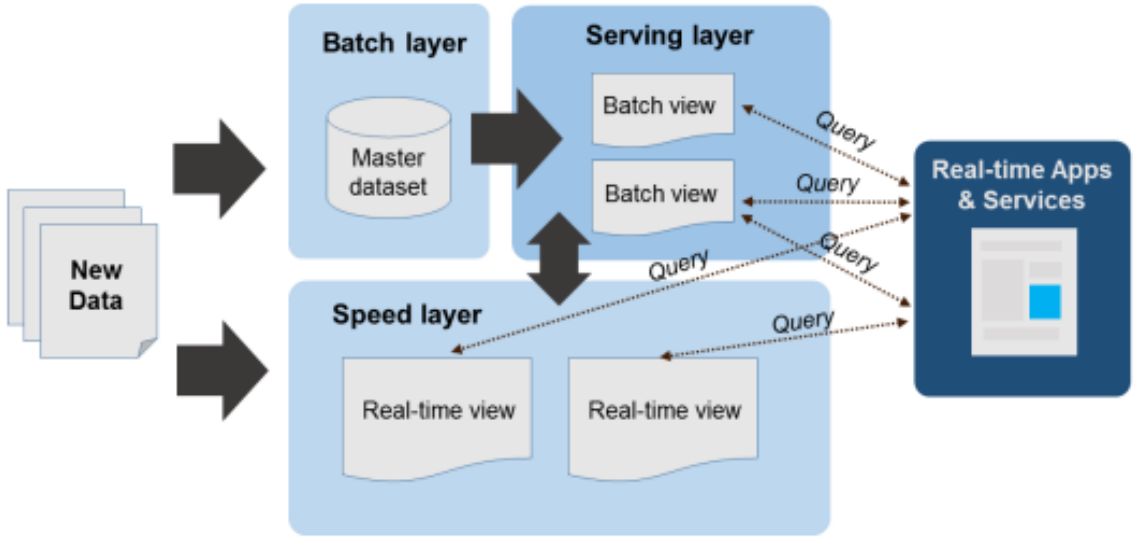
- The Lambda architecture is a data processing architecture designed to handle massive volumes of data in a scalable and fault-tolerant manner. It combines batch processing, real-time/stream processing, and a serving layer to provide a comprehensive solution for big data processing and analytics. The key idea behind the Lambda architecture is to leverage the strengths of both batch and real-time processing to overcome their limitations.



Big Data – Arquitectura Lambda

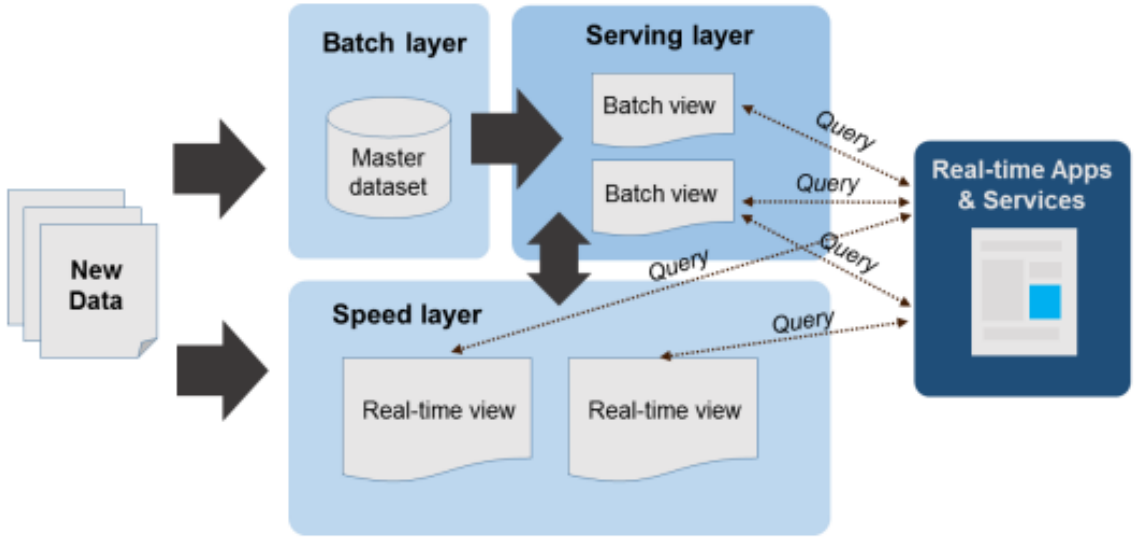
- Batch Layer: This layer handles the batch processing of the entire data set. It takes in the raw input data, performs pre-processing, and generates batch views or data structures. It usually utilizes distributed storage systems like Hadoop Distributed File System (HDFS) and processes data using frameworks like Apache MapReduce or Apache Spark.

- Speed Layer: This layer deals with real-time or near real-time processing of streaming data. It receives data in real-time and performs computations on small time windows or micro-batches. Stream processing frameworks like Apache Kafka, Apache Flink, or Apache Storm are commonly used in this layer.



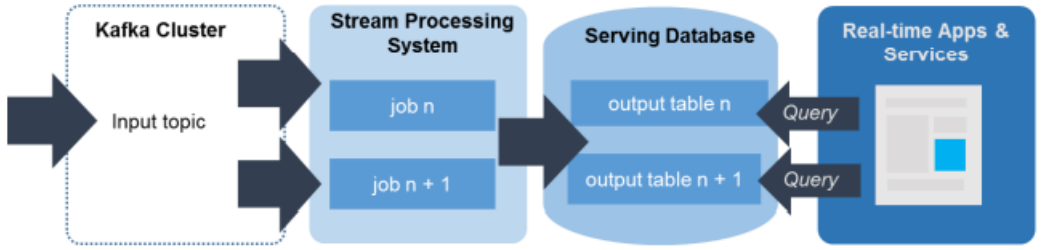
Big Data – Arquitectura Lambda

- Serving Layer: This layer serves the results from both the batch and speed layers to provide a unified view of the processed data. It stores the precomputed batch views and the real-time views and responds to queries from the user interface or applications. Common technologies used in the serving layer include distributed databases like Apache HBase, Apache Cassandra, or Apache Druid.



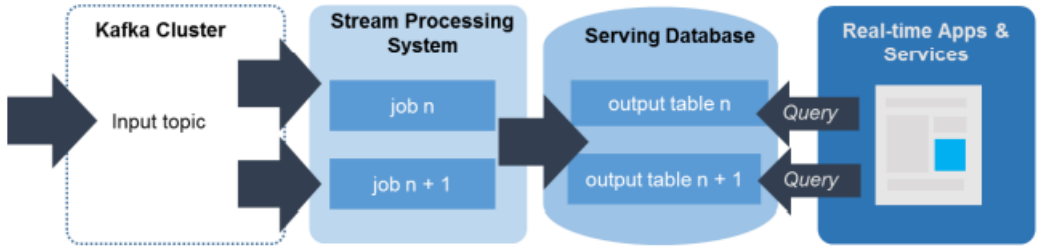
Big Data – Arquitectura Kappa

In the Kappa Architecture, data is ingested in real-time from various sources and processed in a continuous stream. The stream processing system, such as Apache Kafka Streams, Apache Flink, or Apache Spark Streaming, is responsible for processing and analyzing the data as it arrives. It performs transformations, aggregations, and computations on the data in near real-time



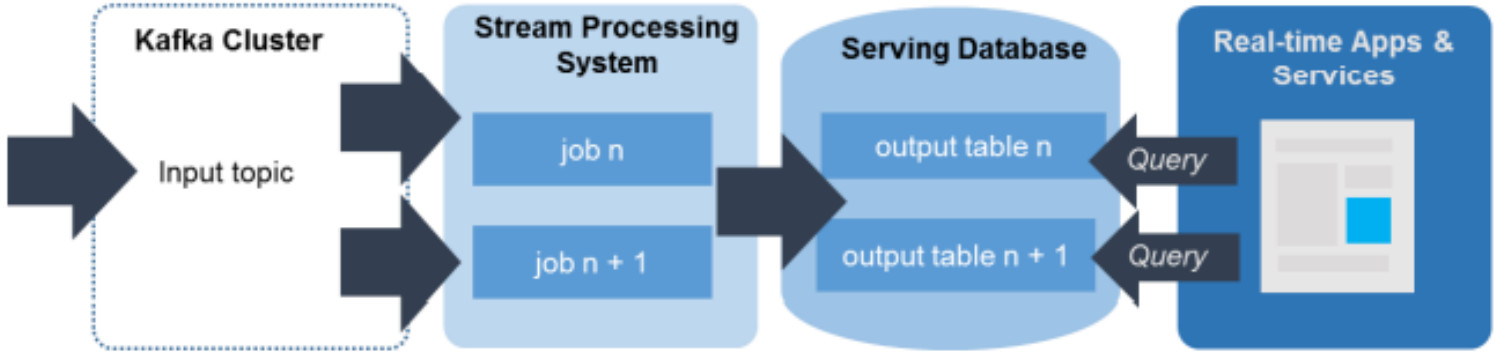
Big Data – Arquitectura Kappa

- Data Ingestion: Data from various sources is ingested into the system in real-time. This can include event streams, logs, sensor data, or any other type of streaming data source.
- Stream Processing: The stream processing system receives the incoming data and performs real-time computations on the stream. It can apply filters, transformations, aggregations, enrichments, and other operations to derive meaningful insights or trigger actions based on the data.



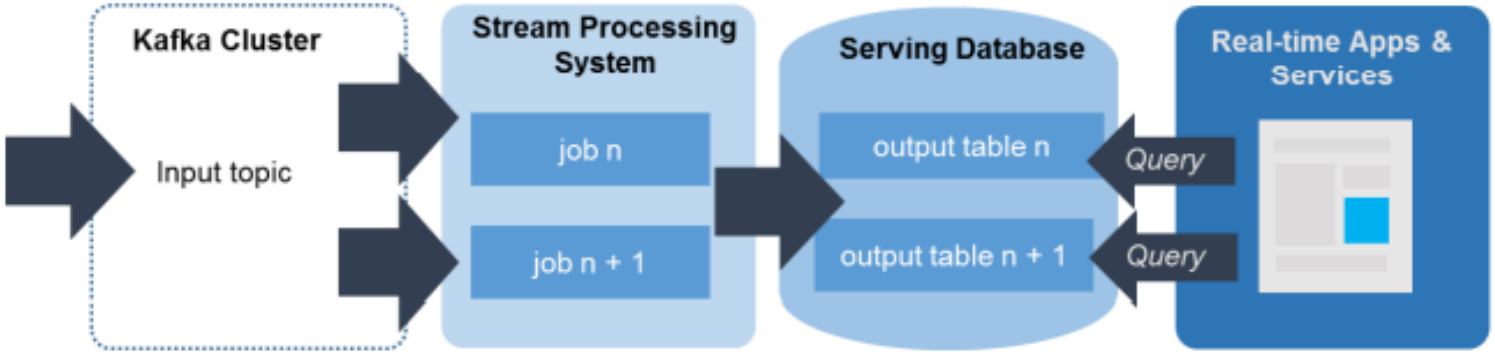
Big Data – Arquitectura Kappa

- Serving Layer: The results of the stream processing are stored in a serving layer that provides access to the processed data. This can be a distributed database or storage allows querying or processed data to or user interfaces.



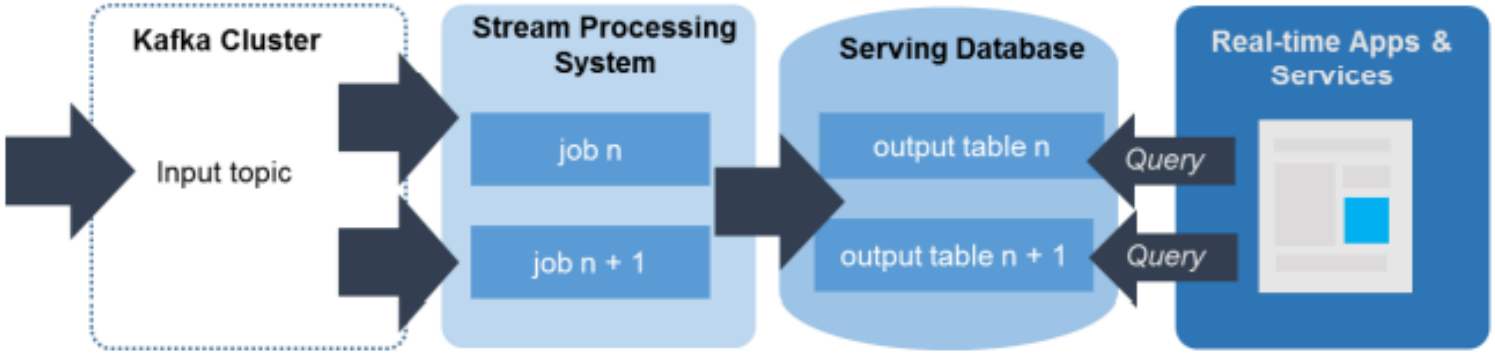
Big Data – Arquitectura Kappa

The Kappa Architecture offers several advantages over the Lambda Architecture. By eliminating the batch layer, it simplifies the overall architecture, reduces latency, and enables faster time-to-insights as data is processed in near real-time. It also eliminates the complexity of managing separate codebases and systems for batch and stream processing.



Big Data – Arquitectura Kappa

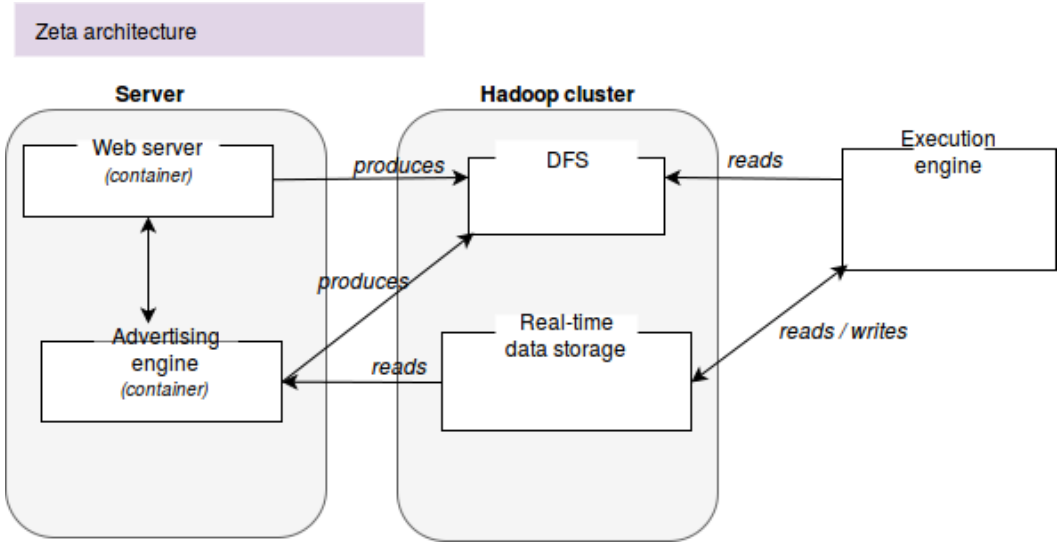
The Kappa Architecture offers several advantages over the Lambda Architecture. By eliminating the batch layer, it simplifies the overall architecture, reduces latency, and enables faster time-to-insights as data is processed in near real-time. It also eliminates the complexity of managing separate codebases and systems for batch and stream processing.



Big Data – Arquitectura Zeta

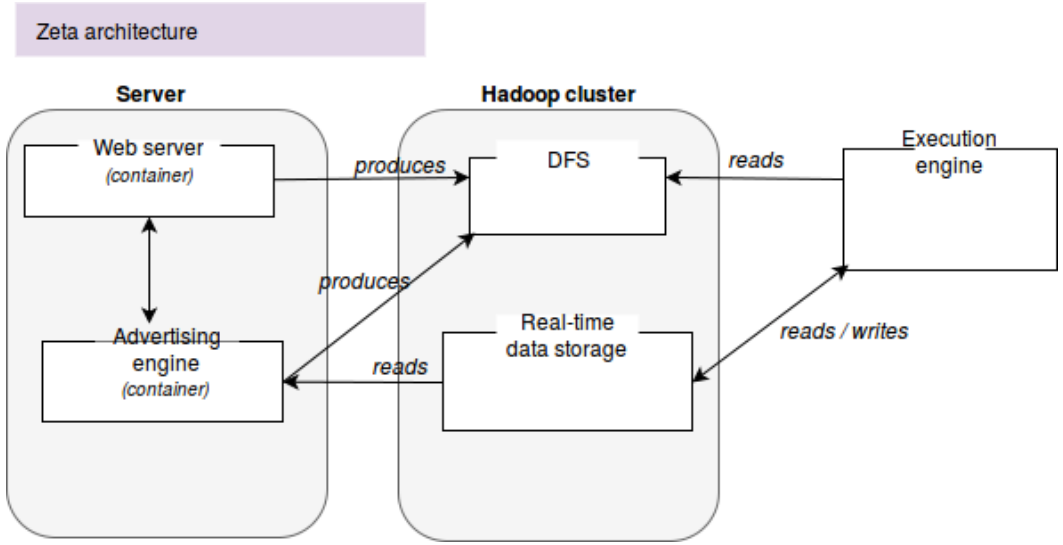
The Zeta Architecture is an alternative approach to big data processing and analytics that combines the strengths of both batch processing and stream processing in a unified architecture. It was introduced by James Warren as an extension of the Lambda Architecture to address some of its limitations.

The Zeta Architecture takes inspiration from the Lambda Architecture but introduces some modifications to improve its flexibility and efficiency. It aims to provide a more balanced approach for handling both historical and real-time data processing.



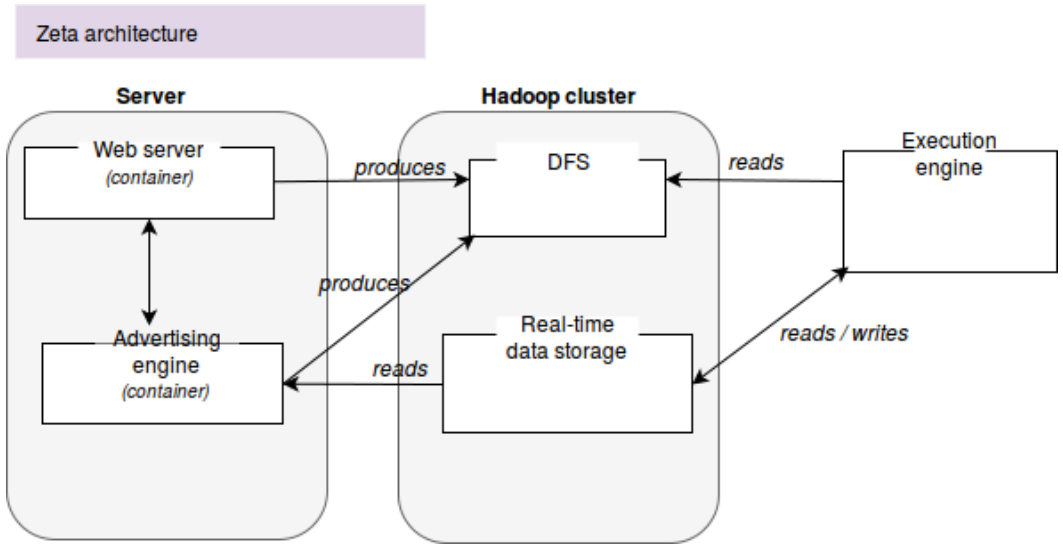
Big Data – Arquitectura Zeta

- Batch Layer: Similar to the Lambda Architecture, the Zeta Architecture includes a batch layer that handles the processing of the entire data set. It ingests the raw input data, performs pre-processing, and generates batch views or data structures. Batch processing frameworks like Apache Hadoop, Apache Spark, or Apache Flink are commonly used in this layer.
- Speed Layer: In the Zeta Architecture, the speed layer is responsible for processing real-time or near real-time data streams. It performs computations on streaming data as it arrives and generates real-time views or summaries. Stream processing frameworks like Apache Kafka Streams, Apache Flink, or Apache Storm can be utilized in this layer.



Big Data – Arquitectura Zeta

- Serving Layer: The serving layer in the Zeta Architecture combines the results from both the batch and speed layers to provide a unified view of the processed data. It stores and serves the precomputed batch views and real-time views, allowing users and applications to query and retrieve the data. Distributed databases like Apache HBase, Apache Cassandra, or Apache Druid are commonly used in the serving layer.



AWS

Herramientas

AWS– S3

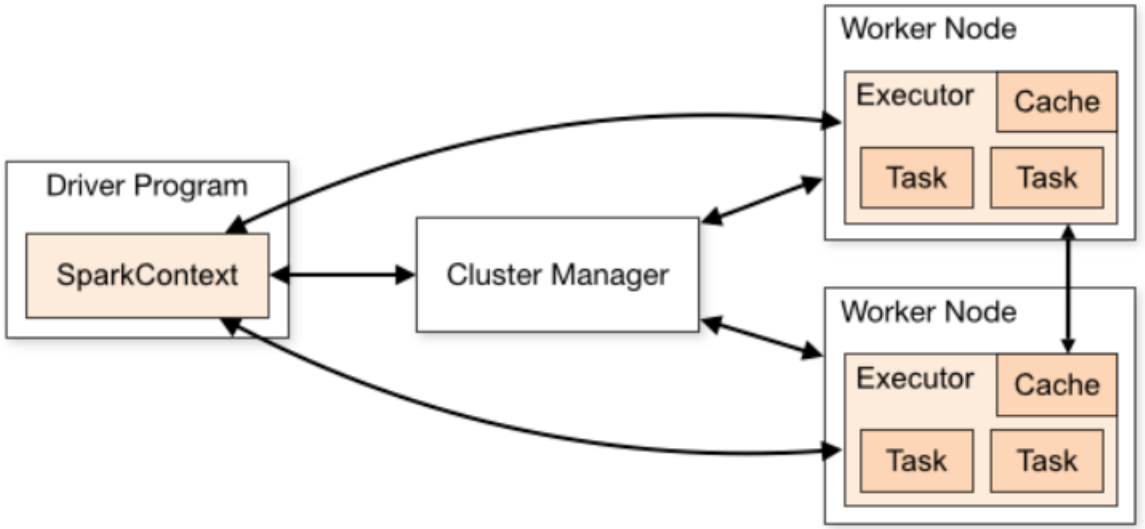
- Object Storage: S3 is an object storage service, which means it stores data as objects. Each object consists of the data itself, a unique key (which acts as the identifier), and metadata associated with the object.
- Scalability and Durability: S3 is highly scalable and can handle virtually unlimited amounts of data. It is designed for 99.999999999% durability, meaning that data stored in S3 is highly resilient and rarely lost.
- Data Accessibility: S3 provides a simple web services interface that allows you to store and retrieve any amount of data from anywhere on the web. It supports both synchronous and asynchronous access methods.

- Security and Permissions: S3 offers several security features to help you protect your data, including encryption at rest and in transit, access control lists (ACLs), and bucket policies. You can also integrate AWS Identity and Access Management (IAM) to manage access and permissions for S3.
- Data Management: S3 provides features for organizing and managing your data, such as versioning, lifecycle policies, and cross-region replication. Versioning allows you to preserve, retrieve, and restore previous versions of an object, while lifecycle policies help you automate the transition of objects to different storage classes based on predefined rules.
- Cost-Effective: S3 offers different storage classes with varying performance and cost characteristics, allowing you to choose the most suitable option based on your data access patterns and budget. It also provides features like intelligent tiering and cost optimization tools to help you minimize storage costs.

AWS– Spark

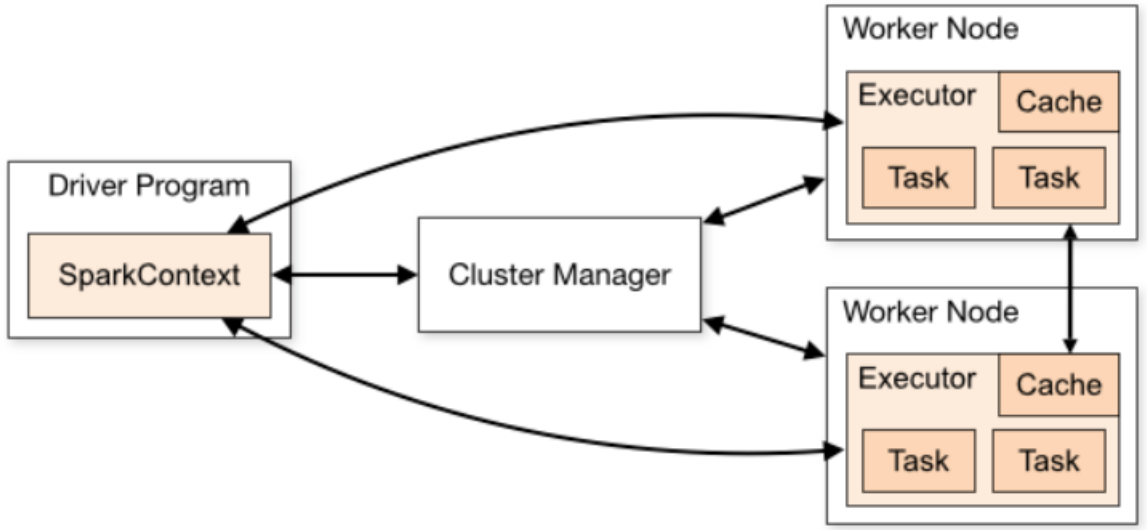
- Each application gets its own executor processes, which stay up for the duration of the whole application and run tasks in multiple threads. This has the benefit of isolating applications from each other, on both the scheduling side (each driver schedules its own tasks) and executor side (tasks from different applications run in different JVMs). However, it also means that data cannot be shared across different Spark applications (instances of SparkContext) without writing it to an external storage system.

- Spark is agnostic to the underlying cluster manager. As long as it can acquire executor processes, and these communicate with each other, it is relatively easy to run it even on a cluster manager that also supports other applications (e.g. Mesos/YARN/Kubernetes).



AWS– Spark

- The driver program must listen for and accept incoming connections from its executors throughout its lifetime (e.g., see spark.driver.port in the network config section). As such, the driver program must be network addressable from the worker nodes.
- Because the driver schedules tasks on the cluster, it should be run close to the worker nodes, preferably on the same local area network. If you'd like to send requests to the cluster remotely, it's better to open an RPC to the driver and have it submit operations from nearby than to run a driver far away from the worker nodes.



Term	Meaning
Application	User program built on Spark. Consists of a *driver program* and *executors* on the cluster.
Application jar	A jar containing the user's Spark application. In some cases users will want to create an "uber jar" containing their application along with its dependencies. The user's jar should never include Hadoop or Spark libraries, however, these will be added at runtime.
Driver program	The process running the main() function of the application and creating the SparkContext
Cluster manager	An external service for acquiring resources on the cluster (e.g. standalone manager, Mesos, YARN, Kubernetes)
Deploy mode	Distinguishes where the driver process runs. In "cluster" mode, the framework launches the driver inside of the cluster. In "client" mode, the submitter launches the driver outside of the cluster.
Worker node	Any node that can run application code in the cluster
Executor	A process launched for an application on a worker node, that runs tasks and keeps data in memory or disk storage across them. Each application has its own executors.
Task	A unit of work that will be sent to one executor
Job	A parallel computation consisting of multiple tasks that gets spawned in response to a Spark action (e.g. `save`, `collect`); you'll see this term used in the driver's logs.
Stage	Each job gets divided into smaller sets of tasks called *stages* that depend on each other (similar to the map and reduce stages in MapReduce); you'll see this term used in the driver's logs.

AWS– Databricks

- Apache Spark: Databricks is built on top of Apache Spark, a fast and scalable data processing framework. Spark provides in-memory processing capabilities and supports various data processing tasks such as batch processing, interactive queries, streaming, and machine learning.
- Collaborative Workspace: Databricks offers a web-based collaborative workspace where teams can collaborate, share code, and document their work. It provides a notebook interface (similar to Jupyter Notebooks) that allows users to write and execute code, visualize data, and create interactive dashboards.
- Data Engineering: Databricks provides tools and features to perform various data engineering tasks, such as data ingestion, data transformation, and data integration. It supports multiple data sources and provides connectors to popular data storage systems like Amazon S3, Azure Blob Storage, and Hadoop Distributed File System (HDFS).
- Data Science and Machine Learning: Databricks enables data scientists to build and deploy machine learning models using popular libraries and frameworks such as TensorFlow, PyTorch, scikit-learn, and MLflow. It provides a rich set of APIs and tools for data exploration, feature engineering, model training, and model deployment.
- Integration and Ecosystem: Databricks integrates with various data sources, data lakes, databases, and data processing tools. It supports integration with cloud platforms like AWS, Azure, and Google Cloud, as well as with popular BI tools and visualization frameworks.
- Scalability and Performance: Databricks leverages the distributed computing capabilities of Apache Spark, allowing it to scale horizontally to process large volumes of data efficiently. It provides auto-scaling capabilities to handle dynamic workloads and supports optimized data caching and partitioning for faster query performance.

Laboratorio - Spark

Your best quote that reflects your approach… “It’s one small step for man, one giant leap for mankind.”

- Neil Armstrong