Justification

Over the last decade, data has seen an exponential growth. According to forecasts, this trend will continue in the foreseeable future as more devices are connecting to internet, generating events that describe the consumer’s behavior on real-time. This has been possible, among other factors, due to the significant drop in the cost of storage and processing.

Businesses around the world are focusing on leveraging data, as it provides a means to discover potential customers, keep current ones interested and swiftly react to market trends. In consequence, many industries are starting to embrace fact-based data-centric decisions to skyrocket their profit.

Provided that more precise and trustworthy predictions demand more information, it becomes crucial to ingest and analyze more data. However, this entails a whole new problem once the data reaches a volume where it is unmanageable by a single machine, since a new level of abstraction is needed: distributed systems.

The problem with distributed systems is that most current open-source technologies require complex tuning and are difficult to integrate with other technologies, in order to create a complete Extraction Transformation and Load (ETL) pipeline. While for large organizations this may be a no-brainer, for small to mid-size ones this becomes extremely difficult to handle, as the expertise required is considerable, experts are sparse, and the implementation cost grows outside the affordable range of the companies, forcing them to utilize inefficient and eventually unmaintainable systems thus limiting their ability to react opportunely.

This work proposes a system that allows users to parsimoniously: (a) migrate already-existing data from a local machine to a cloud data warehouse; (b) create customizable big data pipelines, adapted to the data size of the company and (c) provide a way to graphically visualize aggregates of the data, such as plots and charts.

Relevant concepts

First, let us define what Big Data is about, why it is so important to modern industries and how it differs from traditional data processing mindset. To lay the groundwork, it is important to elaborate on the history milestones to understand the major industry shifts that led to this concept.

# History of data before the XXI century

Prior to 2000, data was exclusively produced and consumed within individual machines. Software installed on PCs collected user behavior data with the only purpose of understanding what actions to take next. In some rare cases, the collected data was utilized to improve user experience. More often than not, data was confined to the individual computer and in most cases died once the computer was thrown away.

Between the late 1990s and early 2000s, internet became accessible to mid-class users as well as widely adopted due to technologies such as email and file transfer protocols such as FTP. A study shows that around 16 million users used to have an internet connection by 1995, which represented the 0.4% of the world’s population. By June 2005, the number increased to a staggering 1018 million users, that represented about 15% of the population [1].

During this period, companies started investing on web sites to lure new customers in by providing new ways of acquiring products on internet. One notorious case is Amazon. Amazon started selling books online on the late 90s, but benefited a lot from the internet growth, as many customers started getting used to this new business model [2].

# New era of connectivity

However, it was not until mid-2000s that connectivity took off with the official presentation of iPhone on 2006. This ushered in a new era: the smartphones one. Smartphones were now available to the masses and the cellular network was advanced enough to handle several thousands of concurrent connections. This in turn contributed to the internet traffic increase and therefore the data production.

# Data Analytics and Business Intelligence

Prior to this data abundance period, company leaders only preoccupied with simple data analysis from their sales data; senior managers commonly based their strategic plans on overly simplistic charts that outlined the company’s sales over a given period of time to predict the income over the next few months. One of the main constrains of such reports was the limited amount information.

Internet offered a solution to the data outage. By having an increasing customer base online, business could collect new data and uncover fascinating trends. This proved to be extremely effective as the companies were able to predict with higher accuracy what a given user would do next or what products they were more interested in.

Data analytics refers to a process to examine data in order to extract value out of it, which translates into predictions that allow organizations to take fact-based decisions and visualize potential risks in the near future. In doing that, they also ensure that both human as well as economic resources are correctly allocated and that the company will be able to react in case of a market shift.

According to [3], Data analysis involves the following major tasks:

1. Defining objectives
2. Posing questions
3. Data collection
4. Data wrangling
5. Drawing conclusions and making predictions

As we will further explore in future sections, data warehousing was pivotal in jumping from simple data analysis to business intelligence. This can sometimes be considered the evolution of the data analysis and involves using more complex technologies, such as ETLs, as well as higher data volume to visualize complex patterns [4].

# SQL and its limitations

Data analysts rely on a number of tools to dissect, massage and visualize data. At their core, most of them use SQL as the main interface to offer a standard interface between the data and the visualizations. SQL stands for Structured Query Language. It has been the de facto domain language to query and aggregate from databases for nearly 5 decades [5].

It allows users to slice and dice a data set using a unique syntax that closely resembles set theory concepts, rather than a typical programming language such as C or Java.

Solutions such as Tableau, Power BI and Qlikview were designed to leverage SQL-like data stores to provide powerful data visualizations for data exploration and wrangling.

Tableau, specifically, is an open-source BI tool to transform data into actionable insights [6]. The tool provides multiple ways to visualize and dissect data on real time, which facilitates the data exploration.

However, SQL was originally designed with individual machines in mind. The underlying data structures that SQL engines utilize are optimized for concurrency in a multi-threaded environment to comply with ACID transactions (Atomicity, Consistency, Isolation and Durability) [7]. Once the data overpasses certain volume, in the order of billions of rows, SQL queries become slow and computationally inefficient.

# Vertical and horizontal scaling

A way to circumvent SQL limitations was to scale up the database, also known as vertical scaling. Vertical scaling implies increasing the computer’s capacity (either the memory size or the storage). This was possible thanks to the evolution of CPUs and the steady transistor size reduction.

Unfortunately, this trend was rapidly coming to an end. Researchers found out that CPU clock frequencies directly influence heat radiation on the device, which eventually will reach the device’s maximum temperature, hence downgrading the microcontroller’s performance [8]. Moreover, despite the huge efforts of hardware designers, the transistor’s size cannot be infinitely reduced; at some point, transistors are so small that quantum effects prevent them from working properly [9]. All this forecasted that vertical scaling would have severe limitations, beside costs, in the near future.

Companies such as Intel and AMD came up with revolutionary CPUs featuring multiple cores within a single chip, as well as optimized CPU cache. This allowed programs to perform operations in true parallelism and effectively fetch frequently used data.

In spite of these improvements, data was still forcing system administrators to look for new long-term solutions to process data efficiently without having to spend millions on it. This is when distributed systems came into play.

Distributed system is a describes many commodity computers interconnected through a network that perform a specific task. This is carried out by dividing the process into much smaller pieces, depending on the data size, so that each computer can work independently on a piece. In the end, the master or coordinator node will take care of orchestrating the whole operation and instruct the slaves to dump the data somewhere [10].

In this way, instead of trying to get a computer with the best specs available on the market, people can simply buy a handful of PCs and connect them. This is known as horizontal scaling or scaling out [10].

# MapReduce

Distributed systems seemed promising, but they added a plethora of new complexities: concurrency, network traffic, data consistency, machine outages, etc. As a result of internal research at Google, a new model emerged to tackle all these nuances: MapReduce. MapReduce is a programming model for processing and generating large data sets in a reliable and scalable manner. Users only have to specify a function (called mapper) to process the key value pair, and then a reducer to collect and aggregate that data [11]. This proved to be highly scalable and work effectively across Google products.

Just around a year later, Google also released the Google File System (GFS) paper which illustrated a fault-tolerant distributed file system for data-intensive applications. MapReduce made extensive use of it to store intermediate results of its computations reliably.

# Big Data

In 2005, Roger Mougalas from O'Reilly Media coined the term Big Data for the first time. It broadly refers to a wide range of large data sets almost impossible to manage and process using traditional data management tools—due to their size, but also their complexity [12].

Over the years, the term has been refined and nowadays scholars as well as industry experts identify 5 characteristics that any Big Data project has in common:

1. **Velocity** – Speed at which data should be processed
2. **Volume** – The amount of data
3. **Value** – The insights that can be extracted from raw data
4. **Variety** – The different formats that can be handled
5. **Veracity** – Trustworthiness or accuracy of the data [10]

# Apache Hadoop

Despite MapReduce’s ingenious design, the technology remained almost unheard, outside of Google, for 2 years more, until a Yahoo employee, Doug Cutting, decided to integrate GFS with MapReduce’s framework [13]. On 2006, Hadoop was born as a framework that allows for distributed processing of nearly unlimited large data sets across clusters of computers. Hadoop is comprised by the following core components:

* **Hadoop Common**: The common utilities that support the other Hadoop modules.
* **Hadoop Distributed File System (HDFS™)**: A distributed file system that provides high-throughput access to application data.
* **Hadoop YARN**: A framework for job scheduling and cluster resource management.
* **Hadoop MapReduce**: A YARN-based system for parallel processing of large data sets [14].

One of the most crucial factors that skyrocketed Hadoop’s popularity and adoption was its open source philosophy, which allowed companies all over the world to play around with it without paying at no cost (other than installing and managing the underlying infrastructure).

Given the above, a rich ecosystem was gradually built around Hadoop by the open source community to support data ingestion, transformation and analysis, as depicted below in the diagram:

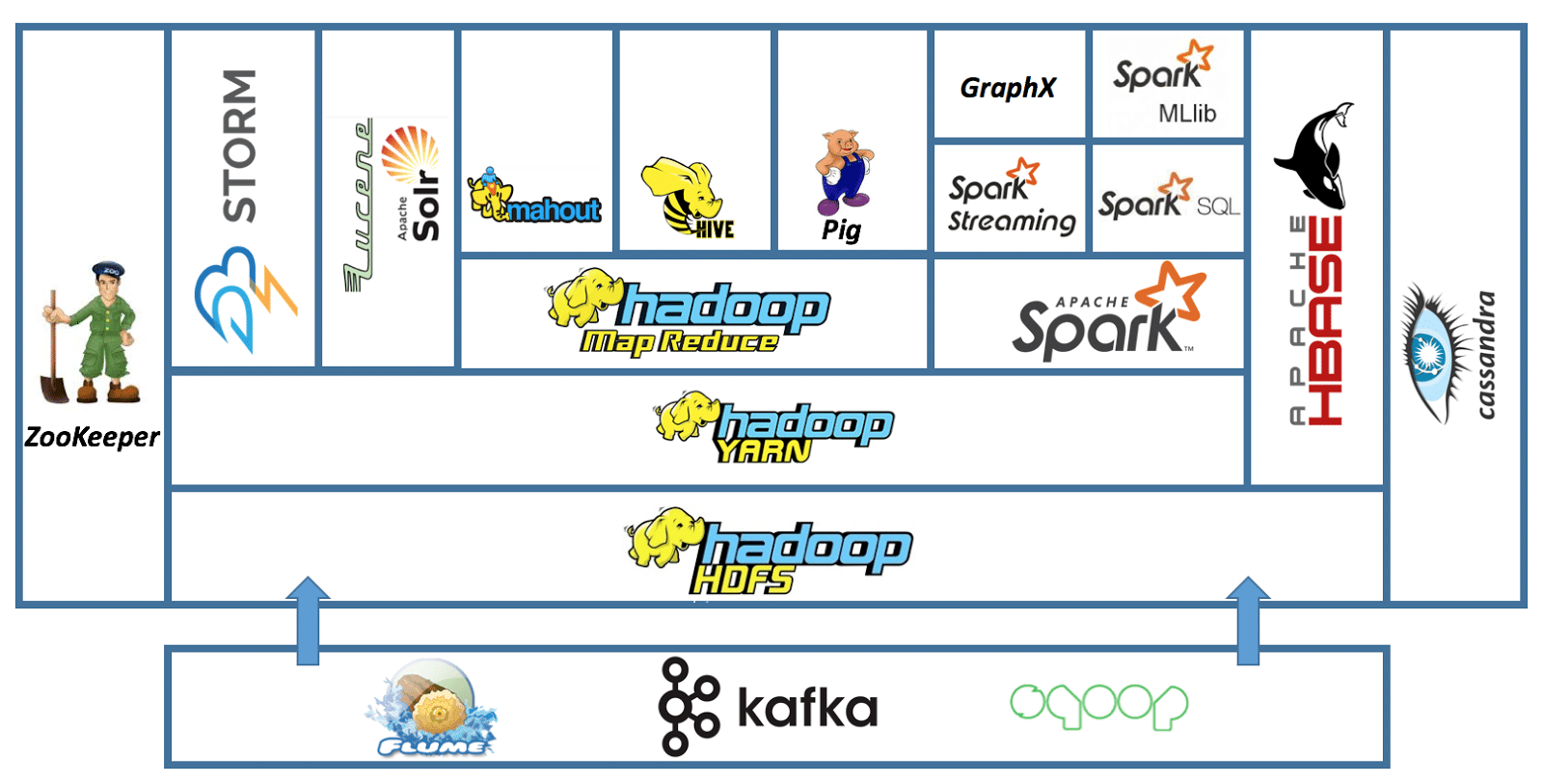


Figure 1 – Hadoop ecosystem [15]

As more industries embraced Hadoop, the complexities of their MapReduce jobs soon became unmanageable. Hence, engineers started struggling to translate already existing and complicated SQL scripts used for their daily analytics to MapReduce, due to the paradigm difference.

In response to this, some companies emerged to try to provide proprietary software to manage Hadoop in an easy manner; companies such as Hortonworks, MapR and Cloudera. In spite of all their efforts, there was still a considerable gap between the usability that companies were looking for and the services these companies could offer.

In 2019, Hortonworks and Cloudera decided to merge to cope with Hadoop’s recent lack of demand [16].

# Apache Hive

Large companies understood that using plain MapReduce to execute distributed tasks was not feasible on the long run. Some of them simply yearned the good old SQL syntax they were used to all along. A particular pain point of using MapReduce is that it was not meant for interactive querying. Facebook was one of the most prominent Hadoop adopters, but as soon as data became unmanageable, they started working on a project able to understand SQL syntax, underpinned by Hadoop. This is how Apache Hive was born [17].

Apache Hive is a data warehouse software that facilitates reading, writing, and managing large datasets residing in distributed storage using SQL [18]. This meant that existing legacy queries could be translated to HiveQL easily to be able to analyze and perform aggregations on the data without the complexity of writing MapReduce.

Hive is primarily focused on data warehouses analytics, since it could perform large file scans and store intermediate results on files. This analytics process was termed OLAP, which stands for Online Analytics Processing.

A huge disadvantage of Hive is that execution of complex queries could take from several minutes to many hours. This because Hive translates SQL queries to MapReduce commands under the hood, which is far from ideal, as many MapReduce operations were unnecessarily writing to disk after each MapReduce stage.

In spite of all this, most users happily migrated to Hive as an intermediate solution.

# Apache Tez

Apache Tez was created as an improvement on Apache Hive. It is an extensible framework for building high performance batch and interactive data processing applications, coordinated by YARN in Apache Hadoop. Tez improves the MapReduce paradigm by dramatically improving its speed, while maintaining MapReduce’s ability to scale to petabytes of data [19]. This is done by analyzing the transformation lineage and creating a Directed Acyclic Graph (DAG), which was analyzed to remove unnecessary MapReduce steps.

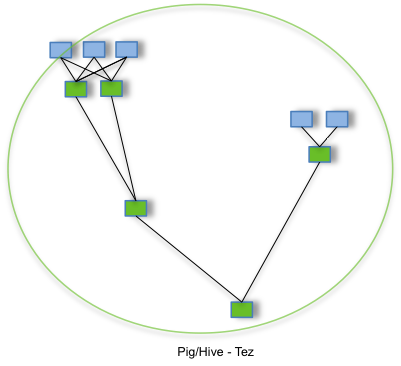
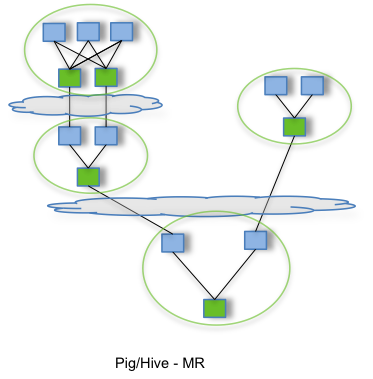


Figure 2 – A MapReduce job lineage (left) and the simplified DAG using Apache TEZ (right) [20].

Queries commonly reduced their execution time from several hours to less than an hour and analysis could be carried out in less time and the computation cost also decreased, as the MapReduce jobs were optimized.

# Apache Spark

As the reader may notice, multiple implementations leaned on MapReduce as it originally promised to handle ever-increasing data volumes. However, as data was exponentially growing, even Apache Hadoop and the afore-mentioned technologies started falling behind. Luckily, in 2010, a new

promising project came into existence: Apache Spark [21].

Apache Spark is a distributed compute engine used to leverage the full parallelism of every node in a cluster in an efficient manner. This was possible thanks to its simple yet clever architecture which analyzed the main architecture pain points of MapReduce and refactored them. Some benchmarks even demonstrate that Apache Spark is up to 100 times faster than a typical MapReduce job on Hadoop [21] [22].

To achieve such efficiency, Apache Sparks relies on in-memory computation. This allows both driver and workers to perform all operations in Random Access Memory (RAM) and keep the results in memory until they need to flush it somewhere. Moreover, by dynamically building a DAG, in a similar fashion to Apache Tez, Spark is capable of keeping track of all operations and hence delete redundant vertices.

Another key feature of Spark is the separation of functions into two main categories: transformations and actions. Transformations specify a number of steps in which the input data will be changed in a particular way; this will not trigger any distributed computation, but rather will accumulate on Spark’s logical plan for future execution, this is called lazy evaluation. Actions on the other hand represent cues for Spark to run the transformations according to the logical plan it assembled along the way.

Spark coins the term Resilient Distributed Dataset (RDD) which, as the name implies, is a distributed collection of structured elements. Since this collection is represented as a typical object, the programmer does not have to worry about complex tuning or any composition to make RDDs work. It works just the same as a collection, with the only difference being that the actual execution takes place on a set of nodes on a cluster rather than in the master [21].

This simplicity allows users to quickly understand data lineage and run complex queries, including merging from a variety of sources and formats, all within a few lines of code.

Latest versions of Spark offer two more sophisticated structures built on top of: Datasets and DataFrames. They allow users to impose data types on all columns of a dataset, support almost native ANSI SQL and offer a more standard interface to read, aggregate and write data [23].

# Lambda architecture

Technologies such as Apache Hadoop and Apache Spark were particularly well suited for large data processing at any given time; i.e., the data is loaded to a given program, a number of transformations occur, and the result is spit out to a distributed data store (e.g., HDFS, S3, GCS). This process is also known as Extract Transform and Load (ETL) [10].

ETLs are often times associated with Batch processing, which in turn deals with historical data on either a daily or hourly basis. The main disadvantage of this approach is that results will always be outdated for as long as the time window is between execution is defined.

Since most businesses are interested in understanding data as soon as possible, a new processing type gained momentum: stream processing. Stream processing provides users with a simple interface to consume data as it arrives and distribute it to a myriad of systems.

Batch and streaming processing pipelines are not mutually exclusive, but rather used together to extract insights from historical data while also being able to take real-time actions on the user interaction with a web page. N. Marz [24] describes this architecture as lambda architecture and is depicted in the following diagram.

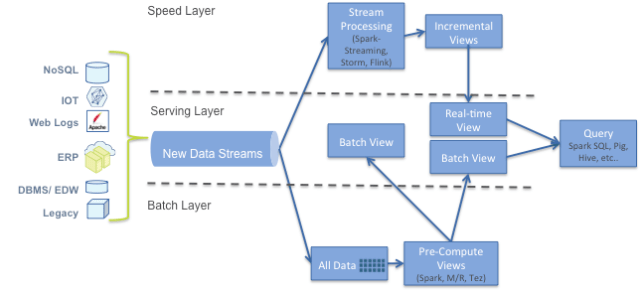


Figure 3 – Lambda architecture

# Apache Kafka

Apache Kafka is a unified distributed streaming platform, with scalability and fault-tolerance in mind [25]. It was originated at LinkedIn and open sourced couple years afterwards.

Kafka has four core APIs:

* The [Producer API](https://kafka.apache.org/documentation.html#producerapi) allows an application to publish a stream of records to one or more Kafka topics.
* The [Consumer API](https://kafka.apache.org/documentation.html#consumerapi) allows an application to subscribe to one or more topics and process the stream of records produced to them.
* The [Streams API](https://kafka.apache.org/documentation/streams) allows an application to act as a *stream processor*, consuming an input stream from one or more topics and producing an output stream to one or more output topics, effectively transforming the input streams to output streams.
* The [Connector API](https://kafka.apache.org/documentation.html#connect) allows building and running reusable producers or consumers that connect Kafka topics to existing applications or data systems. For example, a connector to a relational database might capture every change to a table.

Kafka is generally used for two broad classes of applications:

* Building real-time streaming data pipelines that reliably get data between systems or applications
* Building real-time streaming applications that transform or react to the streams of data

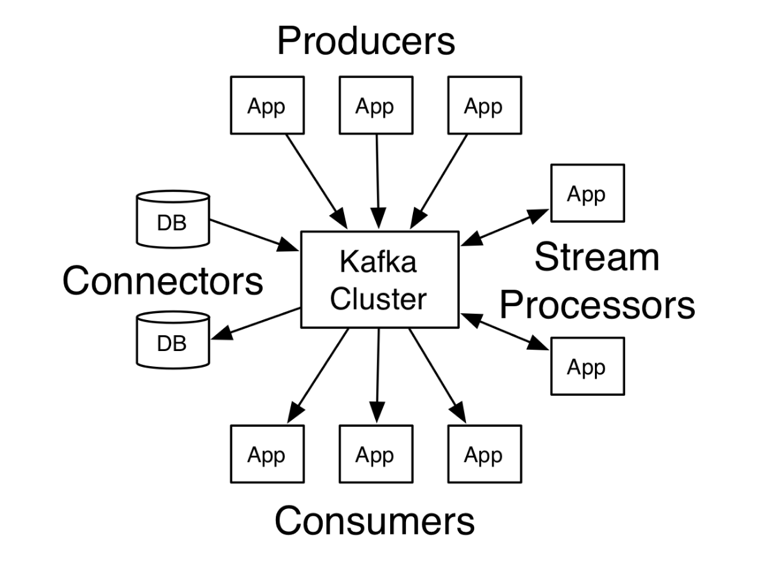


Figure 4 – Apache Kafka high level diagram

One of the main advantages of Apache Kafka is the low latency between the data production and the availability of the data on the given topic (near real time). Furthermore, it keeps track of all messages by default or it can be configured to dump data using the multiple connectors available, which include: Elasticsearch, GCP GCS, AWS S3, syslog, JDBC, Neo4J, etc. [26].

Kafka messages are not limited to text, but they often times convey complex data types, such as lists and maps, represented in JSON format, which allows for schema evolution.

# Spark Streaming

Apache Kafka was paramount for the evolution of Big Data as we know it. Now, more precise analysis was feasible by having fresh data at our disposal. However, stream processing was still not fully available. This is when the Apache Spark community decided to harness the power of Apache Kafka as the entry point for the data and utilize the same semantics as in Apache Spark to create Spark streaming in 2014.

Spark Streaming is an extension of the core Spark API that enables scalable, high-throughput, fault-tolerant stream processing of live data streams. Data can be ingested from many sources like Kafka, Flume, Kinesis, or TCP sockets, and can be processed using complex algorithms expressed with high-level functions like map, reduce, join and window. Finally, processed data can be pushed out to filesystems, databases, and live dashboards. In fact, you can apply Spark’s machine learning and graph processing algorithms on data streams [27].



Figure 5 – Spark streaming internals

Spark streaming groups data into micro batches, which creates the feel of near real time processing. The user can create a handful of aggregations on the DStreams (Spark Streaming abstraction for a distributed collection to process continuously) with Spark’s same syntaxis.

Apache Kafka and Apache Spark Streaming are commonly used hand-in-hand to perform complex transformations. For instance, a Kafka producer can be generating messages from a web log as they come in, whereas in the following section of the pipeline, the Spark Streaming is in charge of chopping the data into pieces (windows) and applying the transformations; after that, the data can be safely stored in a distributed file system.

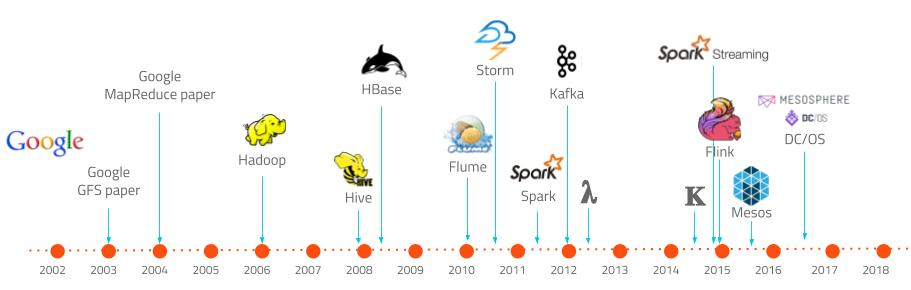


Figure 5 – Big data technologies timeline [28]

# NoSQL

As we have explored in previous sections, SQL has been the cornerstone of many processing tools and data analysis/business intelligence tools. SQL databases are ideal to store information with a well-known schema in a per-row basis. ACID consistency allows users to segregate data in different tables (fact and dimension tables) so that all data is normalized and stored as efficiently as possible.

However, as data started to proliferate, and grow in complexity, thanks to IoT devices, APIs and customer telemetry; SQL proved not to be nimble and flexible enough to accommodate such data. For instance, if a given table required a schema update to incorporate a new column, users would need to update the whole table, even if the prior record did not require so. This generated problems while wrangling the data, as NULL values are the default for the newly created column. In consequence, downstream logic must handle those special values in a graceful way, which often cripples the development of new features in an Agile manner.

To tackle all those limitations, NoSQL philosophy came into existence. NoSQL databases systems are non-relational databases uniquely intended to give high accessibility, reliability, and scalability for enormous data. NoSQL databases can store unstructured data such as email, multimedia, documents, and social media with high performance and very low latencies [29].

NoSQL flexibility allows for schema evolution by not limiting the user to specify it once a collection or a table is created.

As described in [10], NoSQL databases can be divided into four main categories:

1. Key-value stores
2. Columnar or wide-column stores
3. Document databases
4. Graph databases

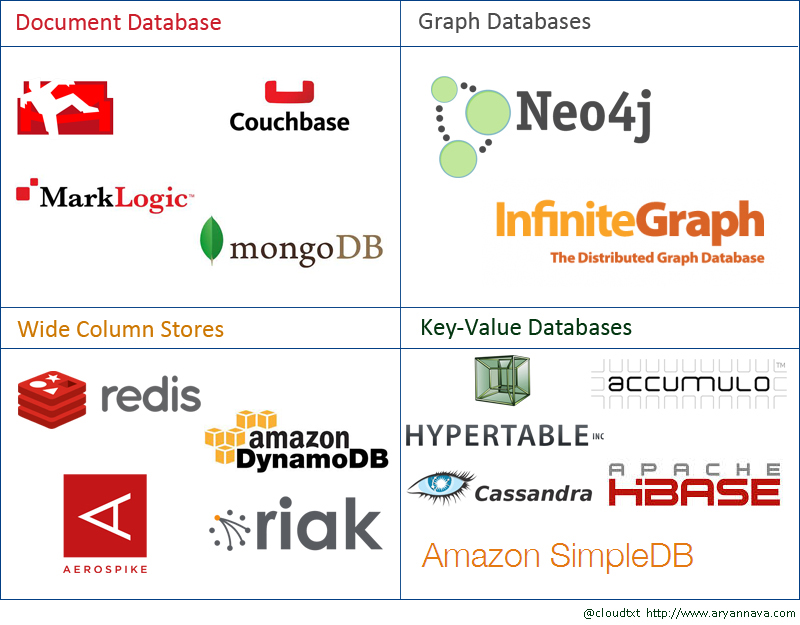


Figure 6 – Different NoSQL databases [30]

Key-value databases are good to represent simple and close-to structured data that can be queried with a unique key. Some examples are: Redis, Riak and Voldemort.

Columnar datastores are used in a similar fashion as SQL databases, since they can be thought of as a table with columns. One key difference is that instead of having NULLs in a column when a record does not have the value, the whole column is missing. This allows to save lots of storage and also improve the read speed. These databases are optimized for large analytics when only a set of columns are required, as they organize same columns within the same file. This allows to scan files quicker and retrieve huge volumes of data efficiently. Also, compression is easier to perform on these records as they are all of the same type within the same column. Some examples are: Cassandra and HBase.

Document databases are somehow similar to key-value databases as they use a unique id to represent a document, however document databases are capable of storing much more complex structures and the documents can be enriched with new fields at any time. This enables users to evolve schemas as soon as new properties are discovered for a particular document. The most prominent example is MongoDB.

Graph databases are the least common type of database used. They represent records with nodes or vertexes on a graph and each record may be related, connected to another via an edge, to a bunch of other records. This type of relationship is found frequently on social media where an individual has multiple friends and each of them have their own set of friends. One example is Neo4J.

Although the use cases for NoSQL are increasing and the popularity is surging, NoSQL is by no means a SQL replacement. In fact, as with most technologies in the Big Data world, NoSQL can go hand-in-hand with SQL datastores, each one storing a specific type of data. Distributed processing tools can then take the data from multiple sources and aggregate it to generate rich reports and visualizations.

# MPP databases

As the adoption of NoSQL databases grew, some companies erroneously began to migrate their legacy databases to NoSQL, under the false impression that this will help them process data quicker. This proved to be a recipe for failure.

NoSQL databases are architected to de-normalize data for optimal query effectiveness while also preserving high resilience and fault tolerance. Hence, data collections should be designed based on a specific business need (e.g., annual reports, monthly reports per user per product, etc.). Because of this, NoSQL databases do not support joins or grouping functions out of the box.

MPP databases were designed as the evolution of SQL databases. MPP stands for Massively Parallel Processing, which means a single computer with many cores interconnected. MPPs have several similarities with clusters, but MPPs have specialized interconnected networks to maximize efficiency [31].

MPPs are capable of processing complex pushdown queries by allocating resources on runtime and executing in parallel. This allows to break down pieces of data and send them to each processor, run the requested transformation, and then collect and collate the results.

Another benefit of MPPs is that they can handle millions of writes and reads almost simultaneously without the need of locks or synchronization logic, as they use a share-nothing schema. In a shared nothing architecture, there is no single point of contention across the system and nodes do not share memory or disk storage. Data is horizontally partitioned across nodes, such that each node has a subset of rows from each table in the database. Each node then processes only the rows on its own disks. Systems based on this architecture can achieve massive scale as there is no single bottleneck to slow down the system [32].

MPPs excel at analytics workloads primarily thanks to the blazing fast processing. This allows analysts to get answers from the data quicker, regardless of the data size.

Everything is controlled by a master node which coordinates the worker nodes. The master node is in charge of managing transactions and keeps track of what is happening on each and every shard. It is capable of constructing the execution plan based off the query.

# Datalake and Datawarehouse

A well-known structure facilitates the understanding of the data, the possibility of mix and to aggregate it with fairly simple logic. At the same time, flexibility allows quicker data ingestion and in consequence the ability to understand unknown variables that may be relevant for future analysis. To handle both scenarios, Datawarehouses and Datalakes are abstractions frequently used.

A Datalake represents a data storage in which all formats are permitted. Usually, a Datalake stores files from different sizes and formats and is commonly replicated across data centers for further fault tolerance. It is the analogous of a local filesystem in which files are the abstraction to represent some piece of information. As the description implies, there is no restriction to what type of data it may contain [10].

Datalakes are very good for data exploration and getting new insights from unexplored data, as it allows to stash in any data and analyze it in the future. As such, Datalakes are often times the starting point for onboarding data from around the organization.

A Datawarehouse, on the other hand, is quite similar to a traditional SQL database. It features a structured or semi-structured schema that represent data that was previously categorized.

Datawarehouses are particularly designed for big queries and analytics that require knowing the data structure upfront. As a result, Datawarehouses are usually some level of transformation from the Datalake into a more insightful source.

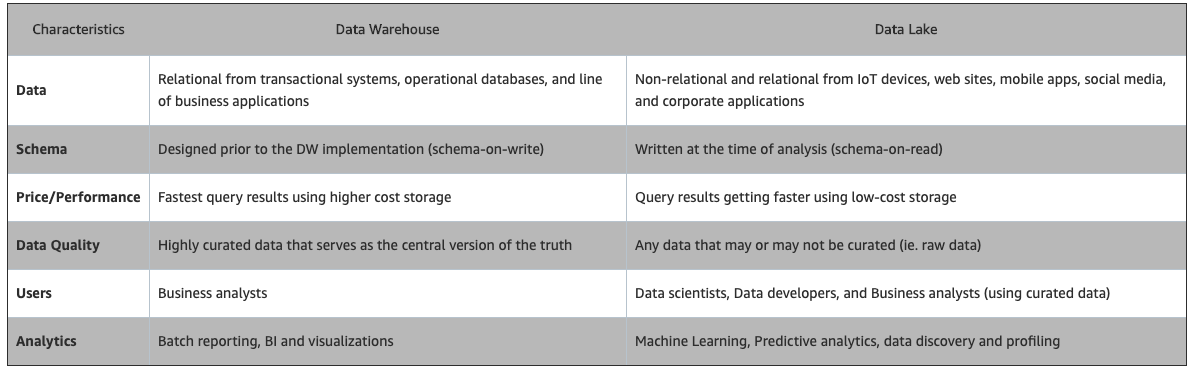


Figure 7 – Data warehouse vs data lake comparison [33]

# ETL and ELT

Data transformation is the bread and butter of Data pipelines. It allows for discovering and understanding hidden patterns that would not simply be possible by looking at all records. In order to extract insights consistently and continuously, scheduled processes need to be created. Such processes are commonly known as ETL and ELT.

Extract, Transform and Load (ETL) describes the process of reading data from an arbitrary data source (e.g., a Data Warehouse, a relational database, files in a distributed object store); applying some aggregation or enrichment such as cleaning, concatenating, grouping, etc.; and storing the result in a destination (similar to the ones observed as data sources) [34].

One of the key considerations while choosing the ideal ETL tool is the data connectors available. A data connector refers to the ability for the tool to read a given format or from a given data source natively. In this case, the more data source the tool supports, the better, as they become critical when an organization wants to read new types of data that were not considered previously.

Performance on large workloads is also crucial aspect of ETLs. As the data volume grows, the ETL tool should automatically scale horizontally to deliver results as soon as possible.

As described by [35], ETL systems add value to the data, as they:

* Remove mistakes and correct missing data
* Provide documented measures of confidence in data
* Capture the flow of transactional data for safekeeping
* Adjust data from multiple sources to be used together
* Structure data to be usable by end-user tools

The downside of ETLs is that, as the complexity of the transformations grows, the latency between the data acquisition and results availability increases considerably. Furthermore, whenever the tool finds an exception while transforming the data, the processing tool will simply fail, and the data will not be stored at all.

Furthermore, ETL processes have to be designed to perform a given transformation on well-known incoming data. As such, it is far from ideal to work with unstructured or unknown data.

Extract, Load and Transform (ELT) processes are a simple yet clever variation of ETLs. ELTs are better suited for Data Lakes, whereas ETLs are almost always used to feed Data Warehouses. ELT process extracts data in the same way as in the ETL approach. Then, the data is loaded into the target Data Lake or Warehouse. Once loaded, the transformations and business logics are applied using native SQL drivers (for Data Warehouses) or in-memory transformations (with Apache Spark, for instance). This helps saving costs by pushing down the transformations to the distributed compute engine and ensures data is checkpointed first, making the whole process fault-tolerant [36].

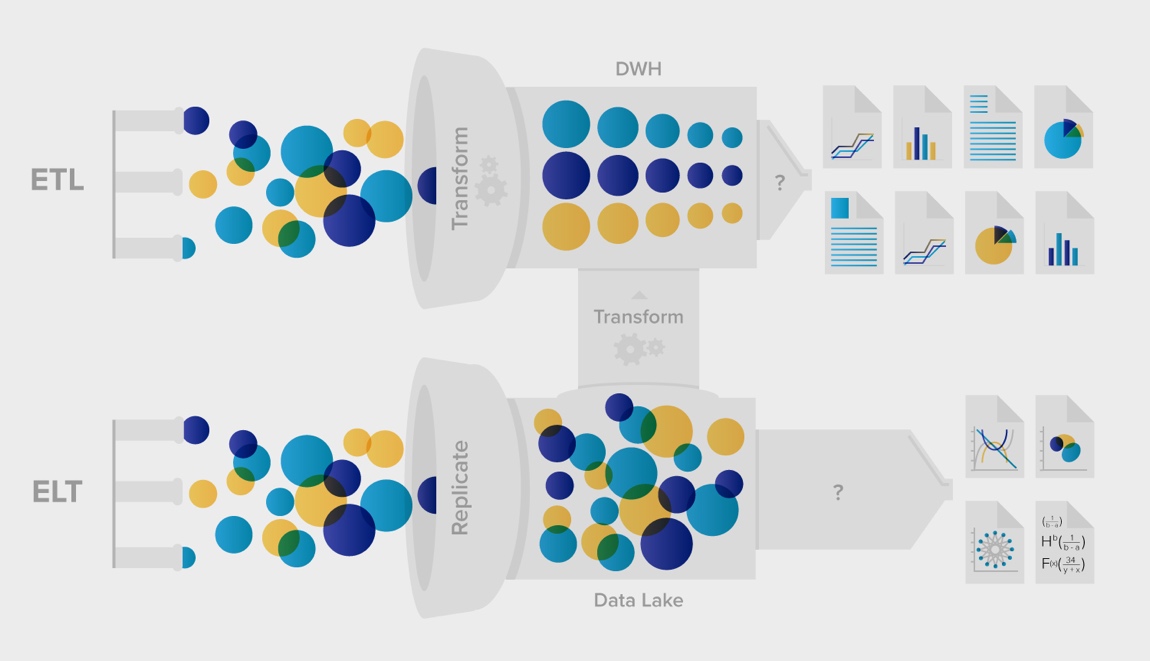


Figure 8 – ETL vs ELT [37]

# Data Governance and Data Lineage

As described in previous sections, a myriad of technologies emerged since Hadoop’s birth to help wrangle and store data in multiple formats, using different processing schemes (batch and streaming).

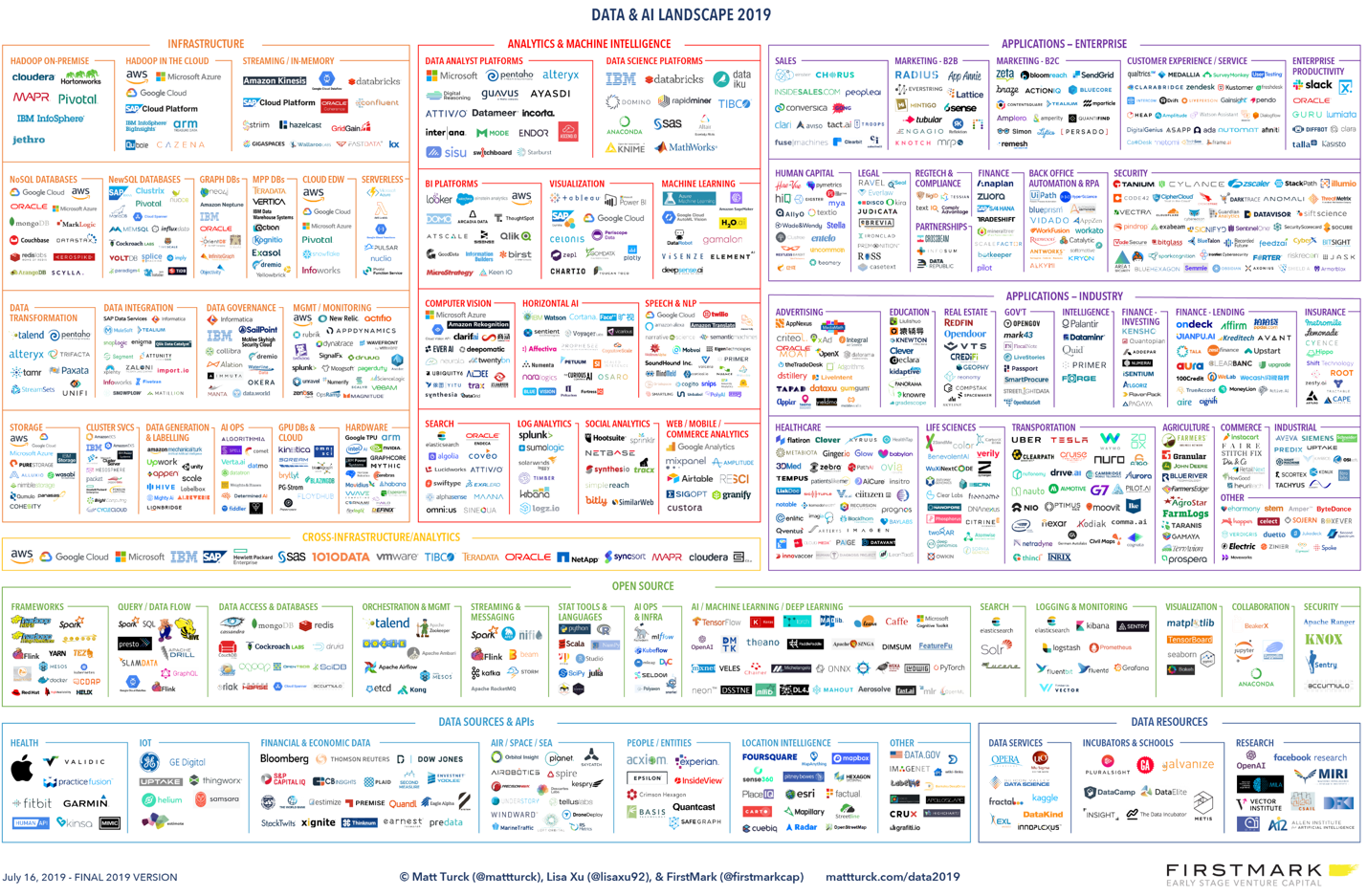


Figure 9 – Big Data Landscape 2019 [38]

Business could finally chain some of these tools together to craft bespoken data pipelines for their exploration and reporting. However, as pipelines grew in complexity, so did the data discovery and lineage: once there were tens of pipelines reading from a dozen different sources, with both structured and unstructured data, and doing complex combinations and enrichments, it virtually became impossible to keep track of any given column, in case some missing data was detected at some point.

To alleviate such limitations, the term Data Governance was coined. Data governance, from a viewpoint of data, enables an organization to create a systematic data standardization, monitoring and management process, so that all data flows are transparent to any authorized user. To establish an effective data governance system, it needs to be set up in connection with corporate governance, IT governance, and ITA / EA from a company-wide perspective [39].

Data governance is a broad concept and involves not only tools, but also a solid methodology to handle data in an efficient yet secure manner. Some of the most important aspects of data governance involve security, catalog, quality and policies, as observed in the following figure:



Figure 10 – Data Governance components [40]

A crucial aspect of data governance is data lineage, also known as data provenance, which can be defined as the data flow control and visibility of all pipelines across the organization. Data lineage helps users drill down into any step of a dataflow process, keep traceability of what transformations are being carried out and what type of data is being utilized for a given output. Literature around data lineage commonly associates it with metadata management, since part of the lineage traceability requires understanding the metadata of a given dataset before doing any processing.

Even though the use of data lineage approaches is a promising way for big data management, the process is rather complex. The challenges include scalability of the lineage store, fault tolerance of the lineage store, accurate capture of lineage for black box operators and many others. These challenges must be addressed carefully and trade-offs between them need to be evaluated to make a realistic design for data lineage capture [41].

# Cloud platforms

Despite the huge efforts of some of the most prestigious organizations to make the Big Data ecosystem flourish, little to no adoption would have been possible without the right infrastructure to underpin elaborated analysis. Unfortunately, companies have found that setting up infrastructure to perform the right analysis on either batch or streaming data is a daunting task.

The time required for businesses to buy and set up servers can take from a couple days all the way to a whole month. Likewise, companies require a huge investment to get the latest hardware, and even so after a couple years, those servers may be obsolete. Finally, managing those servers becomes a titanic task as more servers are bought to handle extra capacity.

On-premise hardware allowed corporations to split their workloads into multiple computers, leveraging the power of newest tools. However, this entailed a new set of problems, as they had to manage the infrastructure by themselves, which involved hiring specialized people to:

* Update the operating systems of all computers
* Keep the important software up to date
* Diagnose whenever a network issue occurred
* Monitor the overall health of the datacenter
* Replace any damaged hardware

To circumvent all of this, cloud computing became extremely popular and is, at the time of writing, one of the most profitable services in the world. Cloud computing refers to a number of on-demand services of compute power, database, storage, applications and other IT resources via internet in a pay-as-you-go pricing model [42].

Around 2006 Amazon finished the first version of what was called Amazon Web Services (AWS) that included a few services by utilizing most of its idle computing capacity could be transformed into a profitable platform [43]. A few years later, big companies such as Microsoft, Google and IBM joined forces to get a portion of the cloud market demand, calling their cloud services Azure, Google Cloud Platform and IBM cloud respectively. Each one of them trying to follow AWS’ steps as close as possible, and ultimately focusing on specific markets.

# With help of this pricing model, companies were now able to save large sums of money by only focusing on their business logic and not having to worry about infrastructure and scalability. Billing by the hour was also a huge advantage, since individuals were able to spin up clusters, run whatever tasks are needed, and shut them down with just a few clicks or API calls. Furthermore, these services offer high availability, top-notch security and benefit from the low latency of the internal networks of the internet providers.

According to [44], cloud service models can be broadly categorized as:

* Software as a Service (SaaS). Cloud user release their applications on a hosting environment, which can be accessed through internet from diverse terminals (e.g. web browser, PDA, etc.). Cloud consumers do not have control over the multi-tenant cloud infrastructure. Cloud consumers' applications are managed in a single virtual environment on the SaaS to leverage optimized amount of resources in terms of availability, speed, security, maintenance and disaster recovery.
* Platform as a Service (PaaS). PaaS is a development platform supporting the full "Software Lifecycle" which allows cloud consumers to develop cloud services and applications (e.g. SaaS) directly on the PaaS cloud. Hence the difference between SaaS and PaaS is that SaaS only hosts completed cloud applications whereas PaaS offers a development platform that hosts both completed and in progress cloud applications.
* Infrastructure as a Service (IaaS). Cloud consumers directly use IT infrastructures (processing, storage, networks, and other fundamental computing resources) provided in the IaaS cloud. Virtualization is extensively used in IaaS cloud in order to integrate/decompose physical resources in an ad-hoc manner to meet growing or shrinking resource demand from cloud consumers. The basic strategy of virtualization is to set up independent virtual machines (VM) that are isolated from both the underlying hardware and other VMs.

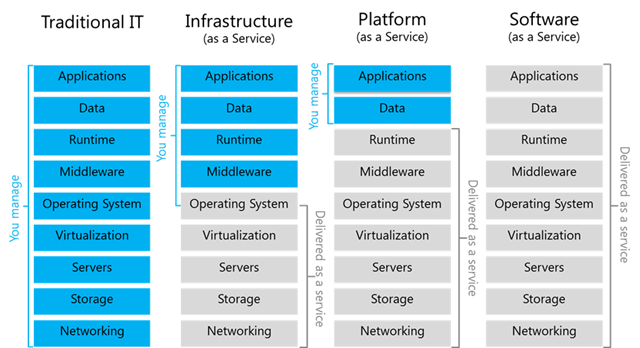


Figure 11 – Cloud service models [45]

Some of the most popular services offered by the majority of cloud providers are:

* Storage
* Compute engine
* Analytics
* ETLs
* Machine Learning platform
* Pub/Sub systems

The only caveat on using cloud computing services is that, because of the nature of pay-as-you-go, companies frequently and mistakenly might leave some services running indiscriminately which will incur in a huge bill at the end of the month. Because of this, cloud engineers may also be required to both quote and supervise the resources usage to avoid unexpected bills.

# DataOps

Data proofed to be one of the most valuable assets for a company, which entailed the creation of many open-source technologies that helped processing and extracting insights from the data. However, as described in the Data Governance section, one of the pain-points that modern organizations are facing is the lack of control and visibility over the data.

Moreover, organizations are not willing to wait many weeks for data analysis to be carried out in a repeatable manner, mainly because of time to market and user churn. To tackle that, a new methodology was born: DataOps.

DataOps can be described as a number of techniques that help data teams to build, maintain and collaborate on pipelines in an effective way. It inherits most of the philosophy from the DevOps practices which establish mechanisms to prevent issues, collaborate effectively across multi-disciplinary teams and speed up deployments through continuous integration and delivery (CI/CD). DataOps additionally incorporates best practices from the Agile methodology, by reducing the time it takes for teams to deliver a given data metric or report and focusing on the Minimum Viable Product (MVP) and working iteratively [46].

# Data engineer, scientist and analyst

Over the recent years, the role of developers and integration teams has shifted significantly with respect to the technologies and the responsibilities each engineer has. With the advent of micro services, for instance, developers were able to deliver higher quality software in a more predictable way and generating less bugs by decoupling large chunks of code into bite-size modules. Once the number of microservices and software products were humongous, it became hard and error-prone to manage infrastructure to deliver on time. This is when DevOps gained momentum.

Site Reliability Engineers, often times known as DevOps Engineers, are usually in charge of isolating the pieces of code in containers (Docker, most of the time) and deploy them on a given cluster. At the same time, developers and analysts had to share responsibilities to be able to handle the data challenges that the companies were facing in order to be able to perform more organic analytics. In consequence the following roles appeared: Data Engineer, Data Scientist and Data Analyst.

A Data Engineer is a developer focused on big data pipelines creation. His/her primary duty is to design, implement, test and improve data pipelines to move data from multiple sources to another while performing some transformation or cleaning. One of the key responsibilities is the optimization of the pipelines on a cloud environment, since the costs of overusing resources may soar unexpectedly if the right considerations are not in place. Data Engineers also participate in introducing new tools or updating previous ones, create jobs to execute ETL or ELT processing on a periodic basis, cleaning dirty data from its raw form, exposing APIs for analysis, and managing the metadata so that subsequent processes can leverage the data without worrying about any missing piece.

A Data Scientist combines parts of the developers and analysts. They are in charge of extracting value of the data. They are particularly good at machine learning and proficient on selecting the most suitable models for a given problem, optimizing and testing hypothesis based on data, as well as generating future predictions. Data Scientists have much more contact with business people than Data Engineers, as they generate proof of concepts for a particular business opportunity.

Data Analysts are those in charge of performing day-to-day analysis to fill in reports and observe common behaviors. They have even more contact with business people. Analysts are specialized in SQL and BI tools to leverage diagnosis and exploratory work. Data Scientists’ discoveries may trigger the incorporation of new analysis into the pipeline, which is performed by the Analysts.

# Data migration

Unless a company was not born in the cloud, they require a method to move data as well as programs or scripts around it to the cloud, when the mere data scale becomes unmanageable or when they decide to scale a product.

Some of the main motivations for migrating to the cloud are the time-to-market for products, as the velocity of product development has increased dramatically, and companies need to quickly finish products to start testing with customers, as well as the disaster recovery for the data in case some local storage gets damage all of a sudden.

Nevertheless, there are a number of challenges to take into account upfront while migrating to a cloud environment:

* Security
* Compliance
* Availability
* Cost

Security – Many companies handle sensitive data from their customers, and the mere fact that such data may be exposed to eavesdropper is scary. Hence, ensuring the data encryption while transporting the data across the wire is crucial.

Compliance – Companies require a way to comply with the wide variety of data formats available and ensure that the tools they use do not hinder their ability to move a given format.

Availability – Since data volumes may be overwhelming, organizations need to be sure that data as well as tools are available most of the time (nowadays above 99.999% of the time).

Cost – Moving to a new environment should not signify that users ought to invest large sums of money. The cost should be low enough to make a profit at some point, otherwise it is meaningless to move to a cloud environment in the first place.

To move data to cloud, engineer require to build programs that connect to cloud services and allow transparent data transfer. Tools such as Apache Kafka are particularly well suited for this task, as it processes messages quickly without sacrificing availability.

State of the Art

Most of what is presented in this work is based upon the work of some of the biggest companies at this time and age. To name a few: Google, Amazon, LinkedIn and Databricks.

MapReduce paper lays the foundations of distributed computing by providing a framework to distribute workloads across a variable number of nodes on a cluster while at the same time offering fault-tolerance and resiliency.

Big Table presented a scalable way of handling data on a distributed system and avoid the most frequent problems a data store may encounter. Also, it introduced a NoSQL data model on which many current storage formats and databases, such as Parquet (DESCRIBE) and Cassandra are based. For example, it first introduced the term ‘column families’ that Cassandra uses to logically group data together, as well as having timestamps embedded for each record to represent the date of creation out of the box.

Dremel describes a system that supports interactive analysis on large datasets on commodity hardware using SQL syntax and scaling to petabyte-size. It reflected the need for analytic tools to explore and aggregate data in a reliable manner. It elaborated more on the columnar representation of records to efficiently store elements on disk, by arranging elements of the same column next to each other, in contrast to record-oriented formats where all columns of a given records are stored consecutively, and this slows down analytic workloads. Finally, it explains the query execution architecture and the enhancements it can bring by working with columnar data.

Dynamo complemented massive data stores by showcasing a different coordination mechanism with a serverless architecture which can achieve much higher availability thanks to its lack of master and replacing it with a quorum of nodes to replicate data. Also, it demonstrates how good this model is at handling failure of multiple nodes at the same time. Finally, it presents the e-commerce architecture at Amazon and how Dynamo fit in the whole picture.

Spark introduced the learning from several engineers while trying to move away from the old MapReduce ideology. It also explored new ways to compute distributed datasets while dramatically improving execution. Thanks to its cutting-edge architecture, it became the most used ETL and ELT tool due to its variety of supported data sources. Finally, it innovated the way it exposes a user-friendly syntax to work with these collections that are easy to understand.

Kafka lays the foundation of distributed messaging systems by offering low latency and high availability. It proposed the usage of brokers to deliver messages and used the terms producer and consumer to refer to both ends of the pipeline, and topic for the pipeline itself. Kafka’s simple architecture to splits topics into partitions was also crucial to deliver to millions of consumers simultaneously without performance impacts to the rest of the users. Finally, it exposed a rich API to deliver messages across systems without worrying about fallback logic to guarantee a message is sent to its destination.

References

[1] <https://www.internetworldstats.com/emarketing.htm>, “Internet growth statistics”

[2] <https://www.thebalancecareers.com/amazon-com-company-research-2071316>, “Overview of Amazon.com's History and Workplace Culture”

[3] <https://www.makeuseof.com/tag/what-is-data-analysis/>, “What Is Data Analysis and Why Is It Important?”

[4] <https://dataconomy.com/2014/07/the-history-of-bi-the-2000s-and-now/>, “THE HISTORY OF BI: THE 2000’S AND NOW”

[5] <https://www.businessnewsdaily.com/5804-what-is-sql.html>, “What is SQL?”

[6] [https://www.tableau.com](https://www.tableau.com/), “Tableau official webpage”

[7] <https://www.ibm.com/support/knowledgecenter/en/SSGMCP_5.4.0/product-overview/acid.html>, “ACID properties of transactions”

[8] Zhu B. ; Electromagnetic radiation study of Intel Dual Die CPU with heatsink, IEEE Xplore, December 2008

[9] <https://www.popularmechanics.com/technology/a23353/1nm-transistor-gate/>, “Scientists Have Made Transistors Smaller Than We Thought Possible”

[10] Designing Data-Intensive Applications; Kleppman M.; 2017.

[11] Dean J., Ghemawat S., et. al.; “MapReduce: Simplified Data Processing on Large Clusters”, 2004

[12] <https://www.researchtrends.com/issue-30-september-2012/the-evolution-of-big-data-as-a-research-and-scientific-topic-overview-of-the-literature/>, “The Evolution of Big Data as a Research and Scientific Topic: Overview of the Literature”

[13] <https://opensource.com/life/14/8/intro-apache-hadoop-big-data>, “An introduction to Apache Hadoop for big data”

[14] <https://hadoop.apache.org/>, “Apache Hadoop”

[15] <https://intellipaat.com/blog/tutorial/hadoop-tutorial/introduction-hadoop/>, “Introduction to Hadoop”

[16] <https://www.cloudera.com/about/news-and-blogs/press-releases/2019-01-03-cloudera-and-hortonworks-complete-planned-merger.html>, “Cloudera and Hortonworks Complete Planned Merger”

[17] <https://www.edupristine.com/blog/introduction-to-hive>, “Introduction to Hive – A Data Warehouse on top of Hadoop”

[18] <https://hive.apache.org/>, “Apache Hive”

[19] <https://www.cloudera.com/products/open-source/apache-hadoop/apache-tez.html>, “Apache Tez”

[20] <https://tez.apache.org/>, “Apache Tez”

[21] Zaharia M., Chowdhury M., et. al.; Spark: Cluster Computing with Working Sets; 2009

[22] <https://spark.apache.org/>, “Apache Spark”

[23] <https://databricks.com/blog/2016/07/14/a-tale-of-three-apache-spark-apis-rdds-dataframes-and-datasets.html>, “A Tale of Three Apache Spark APIs: RDDs vs DataFrames and Datasets”

[24] N, Marz and and J. Warren; Big Data: Principles and best practices of scalable realtime data systems; Manning Publications Co.; 2015.

[25] <https://kafka.apache.org/intro>, “Apache Kafka - Introduction”

[26]<https://www.confluent.io/hub/?utm_medium=sem&utm_source=google&utm_campaign=ch.sem_br.nonbrand_tp.prs_tgt.kafka_mt.xct_rgn.latam_lng.eng_dv.all&utm_term=apache%20kafka%20connectors&creative=367873994816&device=c&placement=&gclid=EAIaIQobChMIgteh0-7V5AIVSL7ACh2eBA_nEAAYASAAEgKwbPD_BwE>, “Apache Kafka connectors”

[27] <https://spark.apache.org/docs/latest/streaming-programming-guide.html>, “Spark Streaming Programming Guide”

[28] <https://en.paradigmadigital.com/dev/from-lambda-to-kappa-evolution-of-big-data-architectures/>, “From Lambda to Kappa: evolution of Big Data architectures”

[29] Kumar J., Garg V.; Security analysis of unstructured data in NOSQL MongoDB database, [2017 International Conference on Computing and Communication Technologies for Smart Nation (IC3TSN)](https://ieeexplore-ieee-org.ezproxy.iteso.mx/xpl/conhome/8275423/proceeding), 2017

[30] <https://www.complexsql.com/difference-between-sql-and-nosql/>, “SQL World”

[31] Li C., Yang J., et. al.; The Distributed Storage System Based on MPP for Mass Data, 2012 IEEE Asia-Pacific Services Computing Conference, 2012

[32] <https://cloudblogs.microsoft.com/sqlserver/2014/07/30/transitioning-from-smp-to-mpp-the-why-and-the-how/>, “Transitioning from SMP to MPP, the why and the how”

[33] <https://aws.amazon.com/big-data/datalakes-and-analytics/what-is-a-data-lake/>, “What is a data lake”

[34] Diouf P., Boly A., et. al.; Variety of data in the ETL processes in the cloud: State of the art; 2018 IEEE International Conference on Innovative Research and Development (ICIRD); May 2018

[35] R. Kimball, J. Caserta, “The Data Warehourse ETL Toolkit”, 2011

[36] V. Ranjan, “A Comparative Study between ETL (Extract-Transform-Load) and E-LT (ExtractLoad-Transform) approach for loading data into a Data Warehouse”, 2019

[37] <https://www.xplenty.com/blog/etl-vs-elt/>, “ETL vs ELT: Top Differences”

[38] <http://mattturck.com/wp-content/uploads/2019/07/2019_Matt_Turck_Big_Data_Landscape_Final_Fullsize.png>, “A Turbulent Year: The 2019 Data & AI Landscape”

[39] Kim H., Cho J., “Data Governance Framework for Big Data Implementation with a Case of Korea”, 2017 IEEE International Congress on Big Data (BigData Congress), 2017

[40] <https://blogs.starcio.com/2018/07/what-is-data-governance.html>, “What is Data Governance? Data practices that address risk and drive opportunities”

[41] Tang M., Shao S., et. al.; “SAC: A System for Big Data Lineage Tracking”, 2019 IEEE 35th International Conference on Data Engineering (ICDE), 2019

[42] <https://aws.amazon.com/what-is-cloud-computing/>, “What is Cloud Computing”

[43] <https://www.dataversity.net/brief-history-cloud-computing/>, “A Brief History of Cloud Computing”

[44] – B. Abbasov, “Cloud Computing: State Of The Art Research Issues”, Conference on Application of Information and Communication Technologies (AICT), 2014

[45] <https://blogs.msdn.microsoft.com/dachou/2018/09/28/cloud-service-models-iaas-paas-saas-diagram/>, “Cloud Service Models (IaaS, PaaS, SaaS) Diagram”, 2018

[46] Cristopher B., Gil B., et. al.; The DataOps Cookbook, 2019

[47] <https://towardsdatascience.com/what-is-the-difference-between-a-data-engineer-and-a-data-scientist-a25a10b91d66#targetText=Data%20engineers%20build%20automated%20systems,analysts%2C%20and%20other%20engineers).>, “What Is The Difference Between A Data Engineer And A Data Scientist”, 2019