What Happened to Hadoop?

Apache Hadoop emerged on the IT scene in 2006 with the promise to provide organizations with the capability to store an unprecedented volume of data using commodity hardware. This promise not only addressed the size of the data sets but also the type of data, such as data generated by IoT devices, sensors, servers, and social media that businesses were increasingly interested in analyzing. The combination of data volume, velocity, and variety was popularly known as Big Data.

Schema-on-read played a vital role in the popularity of Hadoop. Businesses thought they no longer had to worry about the tedious process of defining which tables contained what data and how are they connected to each other — a process that took months and not a single data warehouse query could be executed before it was complete. In this brave new world, businesses could store as much data as they could get their hands on in Hadoop-based repositories known as data lakes and worry about how it is going to be analyzed later.
Data lakes began to appear in enterprises. These data lakes were enabled by commercial Big Data distributions — a number of independent Open Source compute engines supported in a platform that would power the data lake to analyze data in different ways. And on top of that, all of this being Open Source was free to try! What could go wrong?

Schema-on-Read Was a Mistake

As with so many things in life, the features of Hadoop that were touted as its advantages also turned out to be its Achilles heel. First, with the schema-on-write restriction lifted, terabytes of structured and unstructured data began to flow into the data lakes. With Hadoop's data governance framework and capability still being defined, it became increasingly difficult for businesses to determine the contents of their data lake and the lineage of their data. Also, the data was not ready to be consumed. Businesses began to lose faith in the data that was in their data lakes and slowly these data lakes began to turn into data swamps. The "build it and they will come" philosophy of schema-on-read failed.

Hadoop Complexity and Duct-Taped Compute Engines

Second, Hadoop distributions provided a number of Open Source compute engines like Apache Hive, Apache Spark and Apache Kafka to name just a few but this turned out to be a case of too much of a good thing. A case in point — one commercial Hadoop platform consisted of 26 such separate engines. These compute engines were complex to operate and required specialized skills to duct-tape together that were difficult to find in the market.

The Wrong Focus: The Data Lake versus The App

Third and most importantly, data lake projects began to fail because enterprises placed a priority on storing all the enterprise data in a central location with the goal to make this data available to all the developers — an uber data warehouse if you will versus thinking about how the data will impact applications. As a result, Hadoop clusters often became the gateways of enterprise data pipelines that filter, process, and transform data that is then exported to other databases and data marts for reporting downstream and almost never find their way to a real business application in the operating fabric enterprise. As a result, the data lakes end up being a massive set of disparate compute engines, operating on disparate workloads, all sharing the same storage. This is very hard to manage. The resource isolation and management tools in this ecosystem are improving but they still have a way to go. All this complexity— just for reports.

Enterprises, for the most part, were not able to shift their focus away from using their data lakes as inexpensive data repositories and processing pipelines to platforms that consume data and power mission-critical applications. Case in point, Apache Hive, and Apache Spark are among the most widely used compute engines for Hadoop data lakes. Both these engines are used for analytical purposes — either to process SQL-like queries (Hive) or to perform SQL-like data transformations and build predictive models (Spark). These data lake implementations have not focused enough on how to operationally use data in applications.