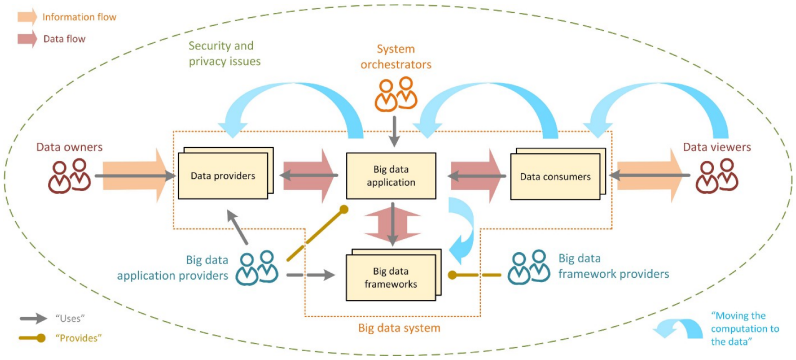
Listen to this story
--:-- 30:53

Everything a Data Scientist Should Know About Data Management*

(*But Was Afraid to Ask)

Phoebe Wong Aug 28, 2019 · 19 min read ★



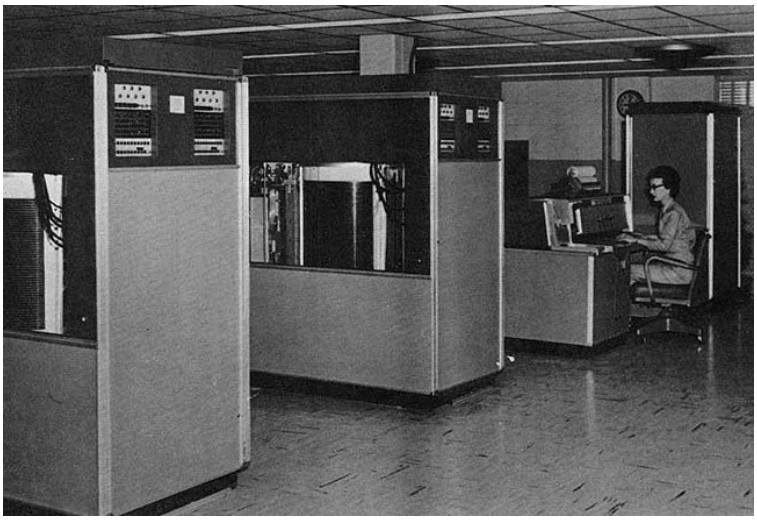
NIST Big Data Taxonomy (Source: WikiCommons)

By Phoebe Wong & Robert Bennett

To be a real "full-stack" data scientist, or what many bloggers and employers call a "unicorn," you've to master every step of the data science process — all the way from storing your data, to putting your finished product (typically a predictive model) in production. But the bulk of data science training focuses on machine/deep learning techniques; data management knowledge is often treated as an afterthought. Data science students usually learn modeling skills with processed and cleaned data in text files stored on their laptop, ignoring how the data sausage is made. Students often don't realize that in industry settings, getting the raw data from various sources to be ready for modeling is usually 80% of the work. And because enterprise projects usually involve a massive amount of data that their local machine is not equipped to handle, the entire modeling process often takes place in the cloud, with most of the applications and databases hosted on servers in data centers elsewhere. Even after the student landed a job as a data scientist, data management often becomes something that a separate data engineering team takes care of. As a result, too many data scientists know too little about data storage and

infrastructure, often to the detriment of their ability to make the right decisions at their jobs. The goal of this article is to provide a roadmap of what a data scientist in 2019 should know about data management — from types of databases, where and how data is stored and processed, to the current commercial options — so the aspiring "unicorns" could dive deeper on their own, or at least learn enough to sound like one at interviews and cocktail parties.

The Rise of Unstructured Data & Big Data Tools

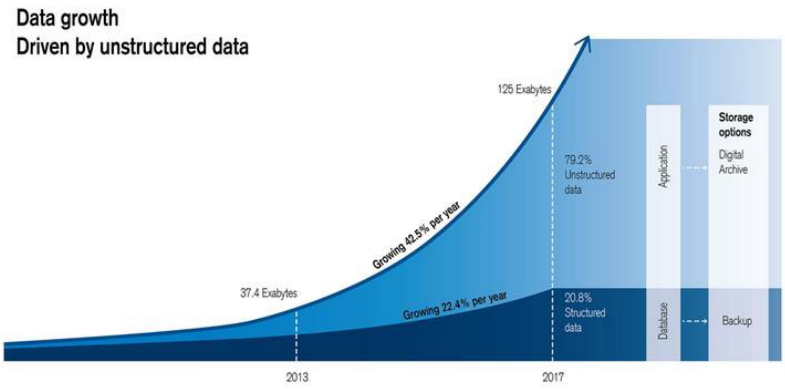


IBM 305 RAMAC (Source: WikiCommons)

The story of data science is really the story of data storage. In the pre-digital age, data was stored in our heads, on clay tablets, or on paper, which made aggregating and analyzing data extremely time-consuming. In 1956, IBM introduced the first commercial computer with a magnetic hard drive, 305 RAMAC. The entire unit required 30 ft x 50 ft of physical space, weighed over a ton, and for $3,200 a month, companies could lease the unit to store up to 5 MB of data. In the 60 years since, prices per gigabyte in DRAM has dropped from a whopping $2.64 billion in 1965 to $4.9 in 2017. Besides being magnitudes cheaper, data storage also became much denser/smaller in size. A disk platter in the 305 RAMAC stored a hundred bits per square inch, compared to over a trillion bits per square inch in a typical disk platter today.

This combination of dramatically reduced cost and size in data storage is what makes today's big data analytics possible. With ultra-low storage cost, building the data science infrastructure to collect and extract insights from huge amount of data became a profitable approach for businesses. And with the profusion of IoT devices that constantly generate and transmit users' data, businesses are collecting data on an ever increasing number of activities, creating a massive amount of high-volume, high-velocity, and high-variety information assets (or the "three Vs of big data"). Most of these activities (e.g. emails, videos, audio, chat messages, social media posts) generate unstructured data, which accounts for almost 80% of total

enterprise data today and is growing twice as fast as structured data in the past decade.



125 Exabytes of enterprise data was stored in 2017; 80% was unstructured data. (Source: Credit Suisse)

This massive data growth dramatically transformed the way data is stored and analyzed, as the traditional tools and approaches were not equipped to handle the "three Vs of big data." New technologies were developed with the ability to handle the ever increasing volume and variety of data, and at a faster speed and lower cost. These new tools also have profound effects on how data scientists do their job — allowing them to monetize the massive data volume by performing analytics and building new applications that were not possible before. Below are the major big data management innovations that we think every data scientist should know about.

Relational Databases & NoSQL

Relational Database Management Systems (RDBMS) emerged in the 1970's to store data as tables with rows and columns, using Structured Query Language (SQL) statements to query and maintain the database. A relational database is basically a collection of tables, each with a schema that rigidly defines the attributes and types of data that they store, as well as keys that identify specific columns or rows to facilitate access. The RDBMS landscape was once ruled by Oracle and IBM, but today many open source options, like MySQL, SQLite, and PostgreSQL are just as popular.

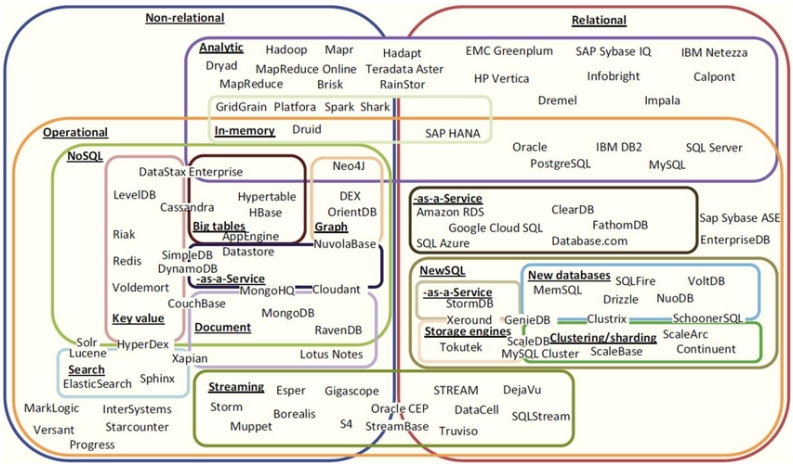
Rank			DBMS	Database Model	Score		
Aug 2019	Jul 2019	Aug 2018			Aug 2019	Jul 2019	Aug 2018
1.	1.	1.	Oracle	Relational, Multi-model	1339.48	+18.22	+27.45
2.	2.	2.	MySQL	Relational, Multi-model	1253.68	+24.16	+46.87
3.	3.	3.	Microsoft SQL Server	Relational, Multi-model	1093.18	+2.35	+20.53
4.	4.	4.	PostgreSQL	Relational, Multi-model	481.33	-1.94	+63.83
5.	5.	5.	IBM Db2	Relational, Multi-model	172.95	-1.19	-8.89
6.	6.	6.	Microsoft Access	Relational	135.33	-1.98	+6.24
7.	7.	7.	SQLite	Relational	122.72	-1.91	+8.99
8.	8.	↑ 9.	MariaDB	Relational, Multi-model	84.95	+0.52	+16.66
9.	9.	↑ 11.	Hive	Relational	81.80	+0.93	+23.86
10.	10.	↓ 8.	Teradata	Relational, Multi-model	76.64	-1.18	-0.77
11.	11.	↑ 12.	FileMaker	Relational	58.02	+0.12	+1.96
12.	12.	↓ 10.	SAP Adaptive Server	Relational	55.86	-0.79	-4.57
13.	13.	13.	SAP HANA	Relational, Multi-model	55.43	-0.11	+3.50
14.	14.	14.	Microsoft Azure SQL Database	Relational, Multi-model	27.99	-0.67	+1.89
15.	15.	15.	Informix	Relational, Multi-model	25.68	-0.18	+0.29
16.	16.	↑ 20.	Google BigQuery	Relational	24.47	+0.55	+10.06
17.	17.	17.	Vertica	Relational, Multi-model	23.80	+0.96	+3.76
18.	↑ 19.	↑ 19.	Amazon Redshift	Relational	22.61	+1.68	+7.43
19.	↑ 20.	↓ 18.	Netezza	Relational	20.84	+0.23	+4.50
20.	↓ 18.	↓ 16.	Firebird	Relational	20.51	-0.88	+0.22
21.	↑ 22.	↑ 24.	dBASE	Relational	16.88	+0.24	+6.75
22.	↓ 21.	22.	Spark SQL	Relational	16.34	-0.41	+3.54
23.	23.	↓ 21.	Impala	Relational, Multi-model	15.07	+0.21	+1.57

24.	24.	23.	Greenplum	Relational, Multi-model	12.77	+0.28	+2.39
25.	25.	25.	Oracle Essbase	Relational	12.41	+0.71	+4.34

RDBMS ranked by popularity (Source: DB-Engines)

Relational databases found a home in the business world due to some very appealing properties. Data integrity is absolutely paramount in relational databases. RDBMS satisfy the requirements of Atomicity, Consistency, Isolation, and Durability (or ACID-compliant) by imposing a number of constraints to ensure that the stored data is reliable and accurate, making them ideal for tracking and storing things like account numbers, orders, and payments. But these constraints come with costly tradeoffs. Because of the schema and type constraints, RDBMS are terrible at storing unstructured or semi-structured data. The rigid schema also makes RDBMS more expensive to set up, maintain and grow. Setting up a RDBMS requires users to have specific use cases in advance; any changes to the schema are usually difficult and time-consuming. In addition, traditional RDBMS were designed to run on a single computer node, which means their speed is significantly slower when processing large volumes of data. Sharding RDBMS in order to scale horizontally while maintaining ACID compliance is also extremely challenging. All these attributes make traditional RDBMS ill-equipped to handle modern big data.

By the mid-2000's, the existing RDBMS could no longer handle the changing needs and exponential growth of a few very successful online businesses, and many non-relational (or NoSQL) databases were developed as a result (here's a story on how Facebook dealt with the limitations of MySQL when their data volume started to grow). Without any known solutions at the time, these online businesses invented new approaches and tools to handle the massive amount of unstructured data they collected: Google created GFS, MapReduce, and BigTable; Amazon created DynamoDB; Yahoo created Hadoop; Facebook created Cassandra and Hive; LinkedIn created Kafka. Some of these businesses open sourced their work; some published research papers detailing their designs, resulting in a proliferation of databases with the new technologies, and NoSQL databases emerged as a major player in the industry.



An explosion of database options since the 2000's. Source: Korflatis et. al (2016)

NoSQL databases are schema agnostic and provide the flexibility needed to store and manipulate large volumes of unstructured and semi-structured data. Users don't need to know what types of data will be stored during set-up, and the system can accommodate changes in data types and schema. Designed to distribute data across different nodes, NoSQL databases are generally more horizontally scalable and fault-tolerant. However, these performance benefits also come with a cost — NoSQL databases are not ACID compliant and data consistency is not guaranteed. They instead provide "eventual consistency": when old data is getting overwritten, they'd return results that are a little wrong temporarily. For example, Google's search engine index can't overwrite its data while people are simultaneously searching a given term, so it doesn't give us the most up-to-date results when we search, but it gives us the latest, best answer it can. While this setup won't work in situations where data consistency is absolutely necessary (such as financial transactions); it's just fine for tasks that require speed rather than pin-point accuracy.

There are now several different categories of NoSQL, each serving some specific purposes. Key-Value Stores, such as Redis, DynamoDB, and Cosmos DB, store only key-value pairs and provide basic functionality for retrieving the value associated with a known key. They work best with a simple database schema and when speed is important. Wide Column Stores, such as Cassandra, Scylla, and HBase, store data in column families or tables, and are built to manage petabytes of data across a massive, distributed system. Document Stores, such as MongoDB and Couchbase, store data in XML or JSON format, with the document name as key and the contents of the document as value. The documents can contain many different value types, and can be nested, making them particularly well-suited to manage semi-structured data across distributed systems. Graph Databases, such as Neo4J and Amazon Neptune, represent data as a network of related nodes or objects in order to facilitate data visualizations and graph analytics. Graph databases are particularly useful for analyzing the relationships between heterogeneous data points, such as in fraud prevention or Facebook's friends graph.

MongoDB is currently the most popular NoSQL database, and has delivered substantial values for some businesses that have been struggling to handle their unstructured data with the traditional RDBMS approach. Here are two industry examples: after MetLife spent *years* trying to build a centralized customer database on a RDBMS that could handle all its insurance products, someone at an internal hackathon built one with MongoDB within hours, which went to production in 90 days. YouGov, a market research firm that collects 5 gigabits of data an hour, saved 70 percent of the storage capacity it formerly used by migrating from RDBMS to MongoDB.

Data Warehouse, Data Lake, & Data Swamp

As data sources continue to grow, performing data analytics with multiple databases became inefficient and costly. One solution called Data Warehouse emerged in the 1980's, which centralizes an enterprise's data from all of its databases. Data Warehouse supports the flow of data from

operational systems to analytics/decision systems by creating a single repository of data from various sources (both internal and external). In most cases, a Data Warehouse is a relational database that stores processed data that is optimized for gathering business insights. It collects data with predetermined structure and schema coming from transactional systems and business applications, and the data is typically used for operational reporting and analysis.

But because data that goes into data warehouses needs to be processed before it gets stored — with today's massive amount of unstructured data, that could take significant time and resources. In response, businesses started maintaining Data Lakes in the 2010's, which store all of an enterprise's structured and unstructured data at any scale. Data Lakes store raw data, and could be set up without having to first define the data structure and schema. Data Lakes allow users to run analytics without having to move the data to a separate analytics system, enabling businesses to gain insights from new sources of data that was not available for analysis before, for instance by building machine learning models using data from log files, click-streams, social media, and IoT devices. By making all of the enterprise data readily available for analysis, data scientists could answer a new set of business questions, or tackle old questions with new data.

Characteristics	Data Warehouse	Data Lake
Data	Relational from transactional systems, operational databases, and line of business applications	Non-relational and relational from IoT devices, web sites, mobile apps, social media, and corporate applications
Schema	Designed prior to the DW implementation (schema-on-write)	Written at the time of analysis (schema-on-read)
Price/Performance	Fastest query results using higher cost storage	Query results getting faster using low-cost storage
Data Quality	Highly curated data that serves as the central version of the truth	Any data that may or may not be curated (ie. raw data)
Users	Business analysts	Data scientists, Data developers, and Business analysts (using curated data)
Analytics	Batch reporting, BI and visualizations	Machine Learning, Predictive analytics, data discovery and profiling

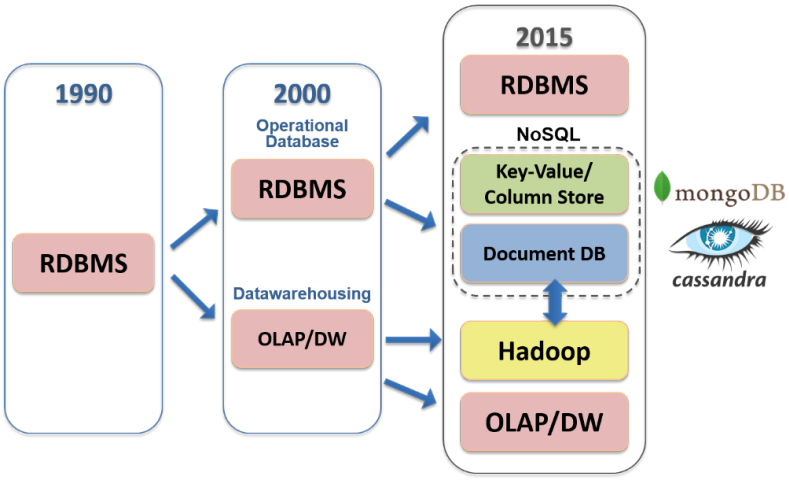
Data Warehouse and Data Lake Comparisons (Source: AWS)

A common challenge with the Data Lake architecture is that without the appropriate data quality and governance framework in place, when terabytes of structured and unstructured data flow into the Data Lakes, it often becomes extremely difficult to sort through their content. The Data Lakes could turn into Data Swamps as the stored data become too messy to be usable. Many organizations are now calling for more data governance and metadata management practices to prevent Data Swamps from forming.

Distributed & Parallel Processing: Hadoop, Spark, & MPP

While storage and computing needs grew by leaps and bounds in the last few decades, traditional hardware has not advanced enough to keep up. Enterprise data no longer fits neatly in standard storage, and the computation power required to handle most big data analytics tasks might take weeks, months, or simply not possible to complete on a standard computer. To overcome this deficiency, many new technologies have evolved to include multiple computers working together, distributing the database to thousands of commodity servers. When a network of computers are connected and work together to accomplish the same task, the computers form a cluster. A cluster can be thought of as a single computer,

but can dramatically improve the performance, availability, and scalability over a single, more powerful machine, and at a lower cost by using commodity hardware. Apache Hadoop is an example of distributed data infrastructures that leverage clusters to store and process massive amounts of data, and what enables the Data Lake architecture.

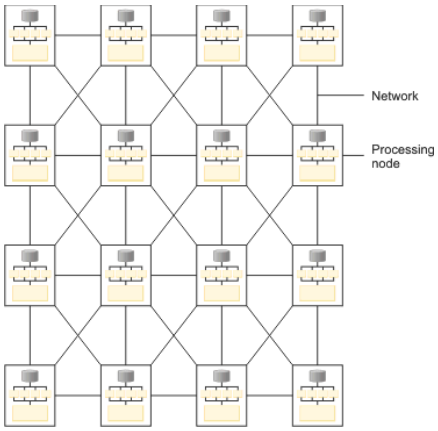


Evolution of database technologies (Source: Business Analytic 3.0)

When you think Hadoop, think "distribution." Hadoop consists of three main components: Hadoop Distributed File System (HDFS), a way to store and keep track of your data across multiple (distributed) physical hard drives; MapReduce, a framework for processing data across distributed processors; and Yet Another Resource Negotiator (YARN), a cluster management framework that orchestrates the distribution of things such as CPU usage, memory, and network bandwidth allocation across distributed computers. Hadoop's processing layer is an especially notable innovation: MapReduce is a two step computational approach for processing large (multi-terabyte or greater) data sets distributed across large clusters of commodity hardware in a reliable, fault-tolerant way. The first step is to distribute your data across multiple computers (Map), with each performing a computation on its slice of the data in parallel. The next step is to combine those results in a pair-wise manner (Reduce). Google published a paper on MapReduce in 2004, which got picked up by Yahoo programmers who implemented it in the open source Apache environment in 2006, providing every business the capability to store an unprecedented volume of data using commodity hardware. Even though there are many open source implementations of the idea, the Google brand name MapReduce has stuck around, kind of like Jacuzzi or Kleenex.

Hadoop is built for iterative computations, scanning massive amounts of data in a single operation from disk, distributing the processing across multiple nodes, and storing the results back on disk. Querying zettabytes of indexed data that would take 4 hours to run in a traditional data warehouse environment could be completed in 10–12 seconds with Hadoop and HBase. Hadoop is typically used to generate complex analytics models or high volume data storage applications such as retrospective and predictive analytics; machine learning and pattern matching; customer segmentation and churn analysis; and active archives.

But MapReduce processes data in batches and is therefore not suitable for processing real-time data. Apache Spark was built in 2012 to fill that gap. Spark is a parallel data processing tool that is optimized for speed and efficiency by processing data in-memory. It operates under the same MapReduce principle, but runs much faster by completing most of the computation in memory and only writing to disk when memory is full or the computation is complete. This in-memory computation allows Spark to "run programs up to 100x faster than Hadoop MapReduce in memory, or 10x faster on disk." However, when the data set is so large that insufficient RAM becomes an issue (usually hundreds of gigabytes or more), Hadoop MapReduce might outperform Spark. Spark also has an extensive set of data analytics libraries covering a wide range of functions: Spark SQL for SQL and structured data; MLib for machine learning, Spark Streaming for stream processing, and GraphX for graph analytics. Since Spark's focus is on computation, it does not come with its own storage system and instead runs on a variety of storage systems such as Amazon S3, Azure Storage, and Hadoop's HDFS.



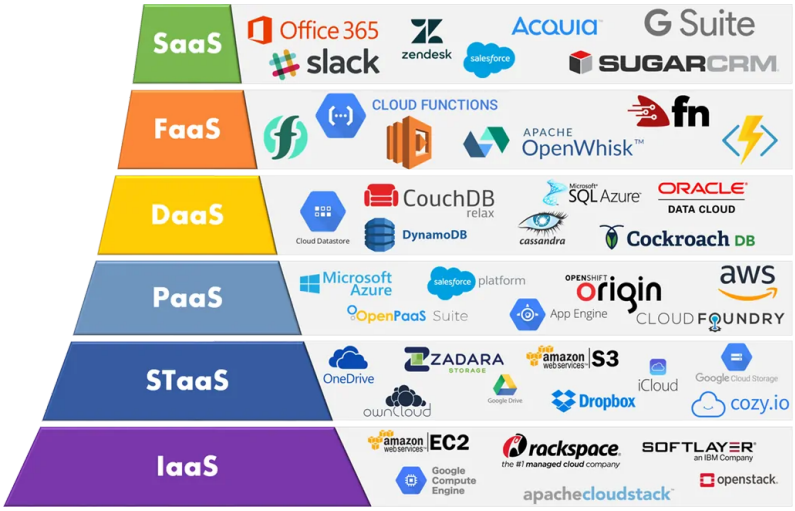
In an MPP system, all the nodes are interconnected and data could be exchanged across the network (Source: IBM)

Hadoop and Spark are not the only technologies that leverage clusters to process large volumes of data. Another popular computational approach to distributed query processing is called Massively Parallel Processing (MPP). Similar to MapReduce, MPP distributes data processing across multiple nodes, and the nodes process the data in parallel for faster speed. But unlike Hadoop, MPP is used in RDBMS and utilizes a "share-nothing" architecture — each node processes its own slice of the data with multi-core processors, making them many times faster than traditional RDBMS. Some MPP databases, like Pivotal Greenplum, have mature machine learning libraries that allow for in-database analytics. However, as with traditional RDBMS, most MPP databases do not support unstructured data, and even structured data will require some processing to fit the MPP infrastructure; therefore it takes additional time and resources to set up the data pipeline for an MPP database. Since MPP databases are ACID-compliant and deliver much faster speed than traditional RDBMS, they are usually employed in high-end enterprise data warehousing solutions such as Amazon Redshift, Pivotal Greenplum, and Snowflake. As an industry example, the New York Stock Exchange receives four to five terabytes of data daily and conducts complex analytics, market surveillance, capacity planning and monitoring.

The company had been using a traditional database that couldn't handle the workload, which took hours to load and had poor query speed. Moving to an MPP database reduced their daily analysis run time by eight hours.

Cloud Services

Another innovation that completely transformed enterprise big data analytics capabilities is the rise of cloud services. In the bad old days before cloud services were available, businesses had to buy on-premises data storage and analytics solutions from software and hardware vendors, usually paying upfront perpetual software license fees and annual hardware maintenance and service fees. On top of those are the costs of power, cooling, security, disaster protection, IT staff, etc, for building and maintaining the on-premises infrastructure. Even when it was technically possible to store and process big data, most businesses found it cost prohibitive to do so at scale. Scaling with on-premises infrastructure also require an extensive design and procurement process, which takes a long time to implement and requires substantial upfront capital. Many potentially valuable data collection and analytics possibilities were ignored as a result.



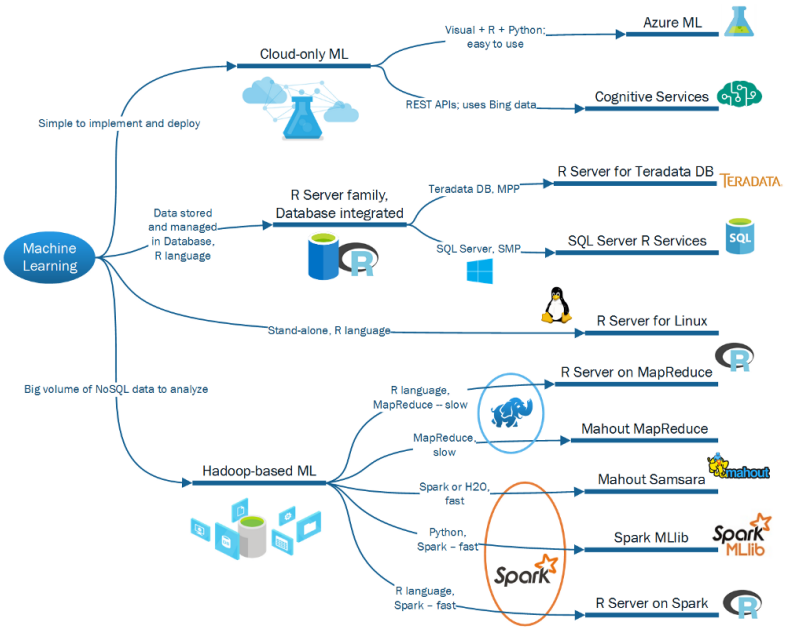
"As a Service" providers: e.g. Infrastructure as a Service (IaaS) and Storage as a Service (STaaS) (Source: IMELGRAT.ME)

The on-premises model began to lose market share quickly when cloud services were introduced in the late 2000's — the global cloud services market has been growing 15% annually in the past decade. Cloud service platforms provide subscriptions to a variety of services (from virtual computing to storage infrastructure to databases), delivered over the internet on a pay-as-you-go basis, offering customers rapid access to flexible and low-cost storage and virtual computing resources. Cloud service providers are responsible for all of their hardware and software purchases and maintenance, and usually have a vast network of servers and support staff to provide reliable services. Many businesses discovered that they could significantly reduce costs and improve operational efficiencies with cloud services, and are able to develop and productionize their products more quickly with the out-of-the-box cloud resources and their built-in scalability. By removing the upfront costs and time commitment to build on-premises infrastructure, cloud services also lower the barriers to

adopt big data tools, and effectively democratized big data analytics for small and med-size businesses.

There are several cloud services models, with public clouds being the most common. In a public cloud, all hardware, software, and other supporting infrastructure are owned and managed by the cloud service provider. Customers share the cloud infrastructure with other "cloud tenants" and access their services through a web browser. A private cloud is often used by organizations with special security needs such as government agencies and financial institutions. In a private cloud, the services and infrastructure are dedicated solely to one organization and are maintained on a private network. The private cloud can be on-premises, or hosted by a third-party service provider elsewhere. Hybrid clouds combine private clouds with public clouds, allowing organizations to reap the advantages of both. In a hybrid cloud, data and applications can move between private and public clouds for greater flexibility: e.g. the public cloud could be used for high-volume, lower-security data, and the private cloud for sensitive, business-critical data like financial reporting. The multi-cloud model involves multiple cloud platforms, each delivers a specific application service. A multi-cloud can be a combination of public, private, and hybrid clouds to achieve the organization's goals. Organizations often choose multi-cloud to suit their particular business, locations, and timing needs, and to avoid vendor lock-in.

Case Study: Building the End-to-End Data Science Infrastructure for a Recommendation App Startup



Machine learning packages for different types of data environment (Source: Kosyakov (2016))

Building out a viable data science product involves much more than just building a machine learning model with scikit-learn, pickling it, and loading it on a server. It requires an understanding of how all the parts of

the enterprise's ecosystem work together, starting with where/how the data flows into the data team, the environment where the data is processed/transformed, the enterprise's conventions for visualizing/presenting data, and how the model output will be converted as input for some other enterprise applications. The main goals involve building a process that will be easy to maintain; where models can be iterated on and the performance is reproducible; and the model's output can be easily understood and visualized for other stakeholders so that they may make better informed business decisions. Achieving those goals require selecting the right tools, as well as an understanding of what others in the industry are doing and the best practices.

Let's illustrate with a scenario: suppose you just got hired as the lead data scientist for a vacation recommendation app startup that is expected to collect hundreds of gigabytes of both structured (customer profiles, temperatures, prices, and transaction records) and unstructured (customers' posts/comments and image files) data from users daily. Your predictive models would need to be retrained with new data weekly and make recommendations instantaneously on demand. Since you expect your app to be a huge hit, your data collection, storage, and analytics capacity would have to be extremely scalable. How would you design your data science process and productionize your models? What are the tools that you'd need to get the job done? Since this is a startup and you are the lead — and perhaps the only — data scientist, it's on you to make these decisions.

First, you'd have to figure out how to set up the data pipeline that takes in the raw data from data sources, processes the data, and feeds the processed data to databases. The ideal data pipeline has low event latency (ability to query data as soon as it's been collected); scalability (able to handle massive amount of data as your product scales); interactive querying (support both batch queries and smaller interactive queries that allow data scientists to explore the tables and schemas); versioning (ability to make changes to the pipeline without bringing down the pipeline and losing data); monitoring (the pipeline should generate alerts when data stops coming in); and testing (ability to test the pipeline without interruptions). Perhaps most importantly, it had better not interfere with daily business operations — e.g. heads will roll if the new model you're testing causes your operational database to grind to a halt. Building and maintaining the data pipeline is usually the responsibility of a data engineer (for more details, this article has an excellent overview on building the data pipeline for startups), but a data scientist should at least be familiar with the process, its limitations, and the tools needed to access the processed data for analysis.

Next, you'd have to decide if you want to set up on-premises infrastructure or use cloud services. For a startup, the top priority is to scale data collection without scaling operational resources. As mentioned earlier, on-premises infrastructure requires huge upfront and maintenance costs, so cloud services tend to be a better option for startups. Cloud services allow scaling to match demand and require minimal maintenance efforts, so that your small team of staff could focus on the product and analytics instead of infrastructure management.

Examples of vendors that provide Hadoop-based solutions (Source: WikiCommons)

In order to choose a cloud service provider, you'd have to first establish the data that you'd need for analytics, and the databases and analytics infrastructure most suitable for those data types. Since there'd be both structured and unstructured data in your analytics pipeline, you might want to set up both a Data Warehouse and a Data Lake. An important thing to consider for data scientists is whether the storage layer supports the big data tools that are needed to build the models, and if the database provides effective in-database analytics. For example, some ML libraries such as Spark's MLlib cannot be used effectively with databases as the main interface for data — the data would have to be unloaded from the database before it can be operated on, which could be extremely time-consuming as data volume grows and might become a bottleneck when you've to retrain your models regularly (thus causing another "heads-rolling" situation).

For data science in the cloud, most cloud providers are working hard to develop their native machine learning capabilities that allow data scientists to build and deploy machine learning models easily with data stored in their own platform (Amazon has SageMaker, Google has BigQuery ML, Microsoft has Azure Machine Learning). But the toolsets are still developing and often incomplete: for example, BigQuery ML currently only support linear regression, binary and multiclass logistic regression, K-means clustering, and TensorFlow model importing. If you decide to use these tools, you'd have to test their capabilities thoroughly to make sure they do what you need them to do.

Another major thing to consider when choosing a cloud provider is vendor-lock in. If you choose a proprietary cloud database solution, you most likely won't be able to access the software or the data in your local environment, and switching vendor would require migrating to a different database, which could be costly. One way to address this problem is to choose vendors that support open source technologies (here's Netflix explaining why they use open source software). Another advantage of using open source technologies is that they tend to attract a larger community of users, meaning it'd be easier for you to hire someone who has the experience and skills to work within your infrastructure. Another way to address the problem is to choose third-party vendors (such as Pivotal Greenplum and Snowflake) that provide cloud database solutions using other major cloud providers as storage backend, which also allows you to store your data in multiple clouds if that fits your startup's needs.

Finally, since you expect the company to grow, you'd have to put in place a robust cloud management practice to secure your cloud and prevent data loss and leakages — such as managing data access and securing interfaces and APIs. You'd also want to implement data governance best practices to maintain data quality and ensure your Data Lake won't turn into a Data Swamp.

As you can see, there's so much more in an enterprise data science project than tuning the hyperparameters in your machine learning models! We hope this high-level overview has gotten you excited to learn more about data management, and maybe pick up a few things to impress the data engineers at the water cooler.

Disclaimer: The views expressed in this article are our own and do not represent the views of our past or current employers.