O'REILLY®

What Is Data Engineering?

A Role for Data Science Enablers

Lewis Gavin

REPORT

What Is Data Engineering?

A Role for Data Science Enablers

Lewis Gavin

Beijing • Boston • Farnham • Sebastopol • Tokyo O'REILLY®

What Is Data Engineering?
by Lewis Gavin

Published by O'Reilly Media, Inc., 1005 Gravenstein Highway North, Sebastopol, CA 95472.

Acquisitions Editor: Jessica Haberman
Development Editor: Amelia Blevins
Production Editor: Christopher Faucher
Copyeditor: Gary Gorski
Proofreader: Christina Edwards
Interior Designer: David Futato
Cover Designer: Karen Montgomery
Illustrator: Jenny Bergman

December 2019: First Edition

Revision History for the First Edition
2019-12-04: First Release

978-1-492-07556-1

[LSI]

Table of Contents

What Is Data Engineering?

Data has increasingly become more popular and is the backbone to many of the modern platforms available today such as Google search, Facebook, Amazon, Spotify, Uber, Tesla, AirBnB, and many more. It's the reason why companies like Google and Facebook are some of the largest in the world. The explosive growth of some of these companies and their ability to transform whole industries, such as the music industry (Spotify) or the taxi industry (Uber), is due to their use of data.

Much of this growth is attributable to the work carried out by data engineers and data scientists. It is common to hear in the mainstream media about data science and the field of machine learning; however, the field of data engineering is just as important and plays a vital role within modern businesses.

Now this isn't a report about the battle of data engineers versus data scientists. Instead, this report will delve into the key aspects of data engineering: why it's an important field based on what it enables and how, together with data science, it provides the backbone to the latest tech in the industry.

Data Engineering Today

Data engineering is becoming more and more popular but is actually a role that has existed for quite some time. Data engineers have been branded in a whole host of different ways in the past, including as database administrators (DBAs), data analysts, business intelligence engineers, database developers, and more.

Recently, although the core role itself has remained fairly constant, the term data engineering has taken on slightly different connotations due to several changes in the market. But at its heart, data engineering is all about the movement, manipulation, and management of data.

This trend has snowballed from the large businesses mentioned in the first paragraph to many others as they come to understand the importance and power of data and not in the traditional sense of just storing basic application and customer data, but storing every event in as little detail as possible. The more data you have, the more knowledge you can redeem from it. You can see in Figure 1 that the global datasphere grew to 40 zettabytes (ZB) in 2019 (1 ZB is around a billion terabytes) and will continue to grow as more businesses harvest its power. This growth in data production has led to major advancements in data technology and data jobs, especially in the fields of data engineering and data science.

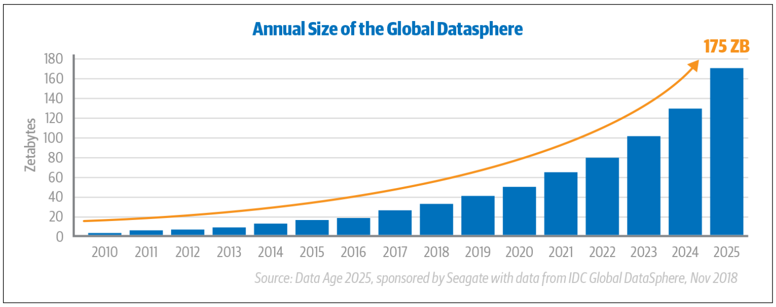

Figure 1. Global data size by year

The existing technologies and previously used infrastructure weren't suited to store this new wave of event data, as the volume became very large, very quickly. This meant that not only would systems eventually slow down but the existing payment models surrounding them meant businesses would have to pay very large amounts of money to support them.

The emergence of "big data" technologies is where data engineering picked up its modern definition. The term big data came into use as early as 1998. John Mashey used it to define the growth in data storage and data production. In 2001 Doug Laney produced a report for Gartner that took this definition one step further and wrapped it up into the "three vees": volume, velocity, and variety.

To deal with the three vees, data warehouses became data lakes and technologies such as Hadoop were developed to help store and process this data on commodity hardware. It seemed like a no brainer for businesses to adopt big data solutions due to the promise of easily scalable storage capacity and faster processing at a much smaller price.

This gave birth to the modern data engineer, a hybrid of a data engineer and a software engineer. Eventually, the marketplace caught up, and Facebook created a technology called Hive for Hadoop that allows SQL to be used. Businesses could then migrate existing skill sets over. Businesses like Facebook and LinkedIn continued to push the boundaries to make the development of big data solutions easier.

However, the differences between the traditional data engineer and a software engineer have become ever smaller as the integration of data capture and data processing has become embedded into core parts of services, instead of just the business information (BI) that sits around them. Finding efficient ways to capture data in real time, process it, generate user-bespoke content, and store it has lead to huge growth in the number of engineers that understand both software and data, and it is now the rule rather than the exception that data engineers have these skill sets.

To leverage all this new data, businesses also needed new methods of processing data. Machine learning and artificial intelligence (AI) became popular very quickly due to the availability of data sets that are large enough to build useful models. Data science is the emerging field for these techniques and is often seen as the money maker over data engineering; however, when the two roles coexist, they complement each other.

Data scientists require data from many sources that need manipulation and aggregation before being used. This is where the marriage with data engineering comes into play, as there is a large amount of engineering required to build an accurate machine learning model. Add on the difficulty of dealing with extra large data sets and these data manipulation tasks quickly become tedious for data scientists, who naturally prefer to (and should) spend their time on increasing the accuracy of their models.

This marriage with data science might explain why data engineering is currently one of the most in demand jobs along with data scientist, with demand for qualified candidates increasing 88% year on year. As seen in Figure 2, Google Trends also shows that interest in the search term “data engineer” is on the rise and is currently at its all-time highest.

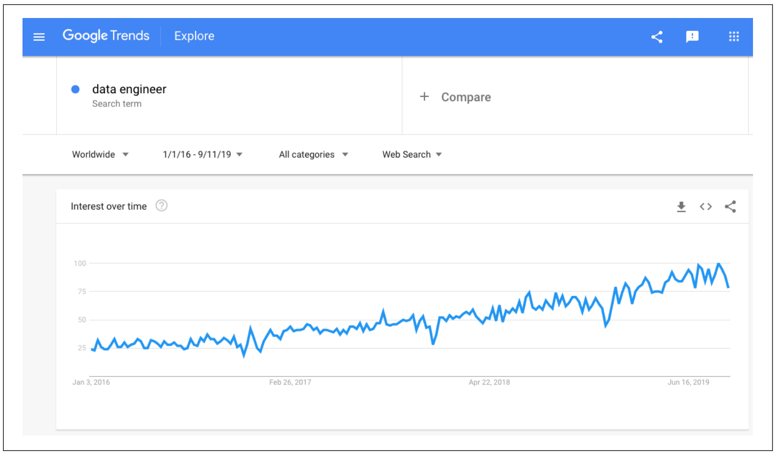

Figure 2. Google Trends graph for search term “data engineer”

Though the data engineer job title doesn’t always come with the same sexy connotations as data scientist, looks aren’t everything and the work that a data engineer does actually makes up a significant portion of the work carried out by a data scientist.

Understandably, topics like machine learning and AI are always going to win a popularity contest, especially as they continue to appear in the mainstream media. However, a good chunk of the work that sits behind these concepts stems from data engineering work, such as data mining, data manipulation and cleaning, string manipulation, aggregation, joining to other data sets, and even building deployable machine learning pipelines. All of these skills are essential to a good data engineer and are necessary when building machine learning models, yet they are less common among data scientists, whose backgrounds are often mathematical. This is why together the two roles are a powerful and complementary combination.

Data Life Cycle

One of the core fundamental practices of data engineering is managing the data life cycle. The life cycle is usually broken down into three key steps: extract, transform, and load, known as ETL, or, more commonly in big data solutions, ELT. The ETL process is shown in Figure 3 and is the basis of pretty much everything a data engineer does. As mentioned, within big data solutions it is common to swap the ordering of the stages around to ELT. I'll talk through each part of the process and their purpose and then explain why swapping the load and transform stages makes sense when working with a big data solution.

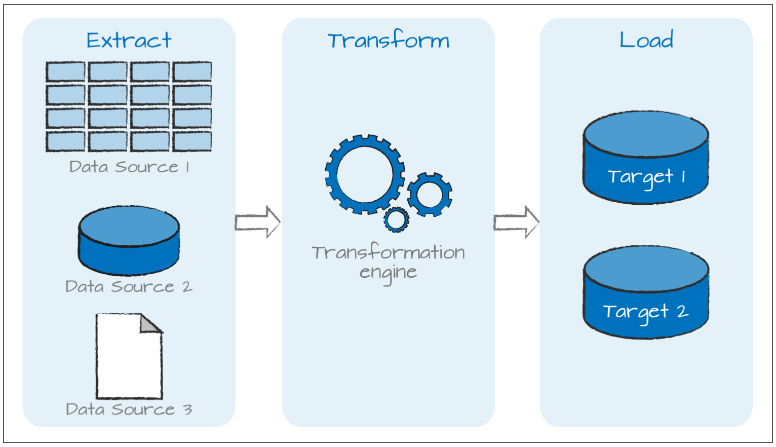

Figure 3. The ETL process

Extract: Getting Data

Talk to any analyst or data scientist and they'll tell you that obtaining data, especially a source that has absolutely everything they require for their job at hand, is near to impossible.

This is why the relationship between analyst or data scientist and engineer is key to building a good model. These roles complement each other well as both have skills that cross over (mainly querying data and simple analysis): the engineers specialize in getting and storing data and analysts and data scientists know how to turn this data into insight. This relationship will be referenced and fleshed out throughout this report.

In the real world, data sets that contain all required data points and are ready for insight are a rare thing and that's where the first skill of the data engineer comes into play. A data engineer dedicates a lot of their time to pulling data sets from an array of sources, most of the time into a central hub, a data warehouse, to be utilized together.

Now, anyone can go and download a static data set from a website. The benefit of a data engineer is just that: the engineering. Not only can they provide a multitude of data from disparate sources, they can automate it so that it's frequently updated, even in real time, if required.

This isn't as straightforward a task as it seems. Data warehouses used to be thought of as spaces for storing key BI and management information (MI) used for reporting on how a business or team is running. This would involve extracting data from the source systems that produce this data, which could be a customer relationship management (CRM) database, for example.

However, the world of data and analytics has moved on and just understanding what is happening isn't enough; businesses want to understand *why* it's happening and even predict what will happen in the future. This is why extraction techniques from disparate data sources are important. You may still load your core business data, but there is much that can be done with third-party data to add value.

These other data sources may not always be stored in a structured way, like within another database, and so they are often considered *dirty* because there's no guarantee of the format. The raw data contained in a data source might be user-generated content or have no obvious delimiters for splitting up the data, web logs often have a variable number of data points and most application programming interface (APIs) provide data in JavaScript Object Notation (JSON) format. A traditional database wouldn't be able to immediately handle this type of data due to tables having strict columns. This is where the "transform" step of the ETL process comes into play.

Transform: Clean, Prep, and Turn Data Into Information

According to a Forbes Survey, 80% of data science work is data prep; 75% of data scientists find this to be the most boring aspect of the job and understandably so. If you've been given a task to build a model to predict which clothing line a user is likely to buy next, yet you spend 80% of your time just getting the data into the right state in order to start building the model, you're going to get frustrated. This will obviously lead to poorer models because you now only have 20% of your time left to build, tweak, and increase the performance of the model.

This is where a data engineer comes into play. The engineer's skills are perfect for this scenario and their addition to the team could flip the 80%/20% split on its head. An experienced data engineer brings their skill in manipulating data to a project, including joining data sets together, fixing null and erroneous values (often done by inferring values by using averages or similar), aggregating data, or manipulating data into features, which is often called feature engineering.

Feature engineering is simply selecting attributes or data points that best describe what you are trying to predict. As a human, your brain does this automatically; it's how we can differentiate between the sky, a ball, a tree, etc. Your brain knows the features of each of these things and uses that knowledge base to decide what an object is (i.e., the sky is usually above you and blue or cloudy, a ball is round and of a certain size, etc.). This may sound simple on the surface but finding the right features and preparing the data to provide these features in a format that is useful for the model is a difficult and often time-consuming task.

However, once the feature selection and cleansing is done, as a freebie, a data engineer can automate the whole process so that as the data is updated and extracted from the source, it's also cleansed and turned into features, providing a consistent source of fresh, clean data.

Let's run through a very crude version of the scenario above.

A data scientist must predict a user's next most likely clothing line purchase. Some obvious data points to start with are user purchase history by clothing line, including amount spent and dates of purchase; demographic data, including age and gender; the number of days between purchases; and seasonal and public holiday data.

Now without a data engineer, they'd have to go to the database and most likely extract a sample data set for a random subset of users. They would need user data and order data and would most likely extract this to a flat file ready to be manipulated in their data science language of choice, let's say Python.

They'd need logic to join the orders and users together so the age and gender data is associated to each sale; however, the first bit of data manipulation is needed at this point to bucket the ages up into even groups. They would then need to add a "days since last sale" column. This should be fairly straightforward, providing each order is time-stamped.

However, they may now want some extra demographic data based on the user's postcode and the seasonal and public holiday details for previous and current years. None of this data is kept in the database as the database only stores data generated by the web application, so they'll need to go and get it. Downloading this data as a one-off should be fairly straightforward once you find a suitable and reliable data source online; however, over time the information will become stale and so will need redownloading. Once the data scientist has downloaded the data set and joined it to the existing data sets, the work can begin to enhance this data into proper features that can be used and understood by the algorithm of choice to build the model.

Now everything mentioned above could be perfectly fine depending on the size of your business and the use case for the model. However, in most cases a data engineer will be able to significantly improve this workflow and remove some of the pain points for the data scientist.

First, if you have a data engineer, you'll more than likely have, or be in the process of building a data warehouse. This means there'll be a single place where your data scientists can retrieve their data. The bonus of this is that the data will be continuously updated at suitable intervals, cleaned and prepared, and stored, ready for use. The logic and calculations performed, such as calculating a user's number of days since last sale, are done in one place and can be reused for other use cases such as business reporting, not just this specific model.

Ad hoc, third-party data sets can also be set up so that they are continuously ingested by either utilizing an available API or other alternative solution. Ultimately, this means that the data scientist can start to work on feature engineering and building the model.

The idea behind this use-case example was to show how important data engineering as a field is in relation to machine learning and AI problem solving. You might find that your data scientists are actually also very good data engineers, in which case they may be happy and able to do the upfront data prep required. If not, building data engineers into the team would be a beneficial step.

Now that we understand common use cases for data transformations and the benefits of automation, it's time to talk about what we should do next with our shiny new data.

Load: Store Data in its Target Location

Load is one of the more basic and less exciting data engineering job functions and that's because it is literally just the movement and storage of data.

In the introduction to the data life cycle, I mentioned that you may sometimes swap the load and transform steps around. Remember that big data technologies such as Hadoop are a lot cheaper to run and spread the processing burden across multiple machines. This often means they can be significantly more powerful than a single standalone machine. In this instance you would want to use this cluster of machines to do all the hard work, as this approach will be more efficient and reinforces why you would want to load your data before transforming. You then perform your transformations on the large data sets using the full power of the cluster.

However, even if you do load into your big data solution first, once it has done all the hard work of transforming the data, you may still want to store it somewhere other than your data lake. Loading traditionally refers to just that—the loading of transformed data into a data storage platform, whether that be in your data warehouse, data mart, or any other downstream system.

As simple as outputting data to a table or file sounds, there are still key data engineering skills here that can help improve the efficiency for the end users. Choosing the correct data storage platform based on the use case is important.

The introduction of cloud computing has simplified the ability to separate storage and computational machines, meaning you can simply scale down (switch off) expensive machines that are used for processing data without affecting the stored data. Once excessive processing is required again, you can simply scale the compute machines back up. This is something to consider when deciding where to load your data.

So let's run through some data storage options that are available both in the cloud and on-premise, including NoSQL databases, relational data stores, and data warehouses.

NoSQL

Interest in NoSQL peaked in 2010 and has been fairly consistent since then as seen in Figure 4. This is due to the architecture naturally allowing distributed computing, low maintenance, and less downtime, each of which I'll discuss in more detail shortly.

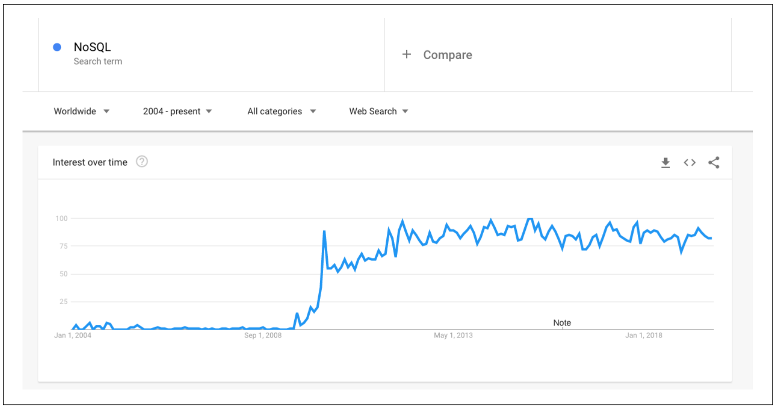

Figure 4. Google Trend screenshot for the search term "NoSQL"

NoSQL stands for not only SQL and refers to a group of technologies that don't tend to store data in tables and columns like a traditional database and do not use SQL to retrieve or access the data.

Google's Bigtable is a great example of a NoSQL data store. Google was after a resilient, distributed, and flexible data storage mechanism that traditional databases couldn't give them, so they came up with BigTable that is built on top of the Google File System (GFS).

As with many things, Google was a pioneer and many others embraced this same concept, opening the way to many other NoSQL-style data stores. NoSQL databases work well when you have something that wouldn't easily be stored in a standard table, such as documents or semistructured data. Large and unstructured data has become more popular due to the internet of things (IoT), social networks, and the rise of AI (due to the human language being a great learning resource for machines).

NoSQL databases have also been popular due to the rise in distributed computing. Traditionally, users were limited to terabytes of data for reasonable costs, but as distributed technologies and file systems (including Hadoop) became more accessible, storing petabytes of data is now a requirement and therefore horizontal scaling of your data storage mechanisms becomes important. A network of machines is called a cluster and each machine is referred to as a node.

Companies also want to reduce downtime. Relational databases could require downtime for things like updating a schema, which is not very efficient, especially if you have a very large table and you want to add a new column to capture some new data. NoSQL databases allow for dynamic schemas, meaning columns can be added freely without any impact and because NoSQL systems are usually distributed and work across a cluster; if a node needs to be taken down for maintenance, the data should be available on other nodes, limiting downtime.

Relational data stores

These data storage options will be the ones most of us are comfortable with, as they've been around for a long time. Many businesses have been using Oracle, MySQL, Postgres, SQL Server, and many others to store their data and even as data warehouses or data marts that are used for BI reporting.

This rich history is one of its main advantages as the technology is well understood, tried, and tested, and the interface, SQL, is a very simple language that is known by a huge number of people.

The ability to index and store data means retrieval of a single row or of multiple rows of data is efficient (up to a certain size anyway), and you have the benefit of key set theory functionality when dealing with data sets, including unions and joins, making it simple to bring data sets together.

Data warehouses and data lakes

As I mentioned in the previous section, businesses have been using relational data stores for data warehousing for years as the point of the data warehouse is to marry data sets together to gain a broader understanding of the relevant business metrics.

In the world of big data none of this has changed. Data warehousing appears different mainly because the storage techniques are slightly different, but at its core you're still going to need to extract, transform, and load data for reporting or machine learning models or anything else.

The key difference is flipping the concept of a single database on its side. Traditionally, if your database needed more power or more storage, you "vertically" scaled it by adding more or better processors and RAM and more or larger hard drives. Modern solutions including Hadoop, Google's Bigtable, or Amazon's Redshift all scale "horizontally," meaning that if you want more power or storage, you just add more machines, distributing the power and storage.

A real-world example to put this in perspective would be a simple visit to a grocery store. Going by yourself to grab a few items is no problem; your items fit in a bag and you can carry them home easily. Now if you needed to buy one hundred items and you went alone, your only solution is to get a much larger bag, but carrying it would be very difficult. You could split the order into multiple bags, but then you wouldn't be able to carry all of them. You could get a stronger person to help, but as you add more items, you will eventually reach an upper limit. Essentially, you have a physical limit on how much you can carry as one person, and at a certain size scaling vertically just doesn't work any more.

The big data paradigm of scaling horizontally is much more effective, as you would essentially go to the store with more people and split the groceries into more bags. Even if you had some people that were stronger than others, you could split the bags proportionally to allow for this. If you reach a limit where you have more shopping than the people can carry, you can just add another person. This is how Hadoop, Redshift, and BigTable solve the scaling problem.

Now, I know what you're thinking: surely at some point scaling horizontally must be more expensive. Yes, there is definitely a tipping point when considering whether to scale three machines vertically versus adding a fourth machine. A data engineer should be able to assess the usage of the cluster, find out where the performance bottlenecks are within data pipelines, and help decide on which scaling option is best.

As you can see, there are many options and each requires slightly different knowledge to get the most out of the chosen technology. This is why a data engineer is a key asset in a modern data-savvy business. A business could have any combination of the above technologies and will undoubtedly need a data engineer to help maintain, integrate, and get the best performance out of each of them.

Real Time

With data flowing and updating from ever-more sources, businesses are looking to quickly translate this data into information to make key decisions, usually in an automated way. This is a perfect scenario for a data engineer's skills, and any business looking for real-time insight should seriously consider hiring data engineers for this task.

Real time refers to the collection of data in real time or as close to real time as possible, almost removing the time between data creation and data collection.

Understanding the benefits of real time is important, though it can be overkill in many use cases. A data engineer should be fully aware of these benefits and be familiar with how and when to use the technologies and architectures discussed in this section. Real-time data streams have become more popular again due to the IOT, sensors in everyday devices, and of course the rise of social media. These platforms provide updates on ever-changing states, and users expect to see feedback from their actions in real time. Processing and analyzing this data, even a day later, can give misleading and now currently false information. Imagine hitting "Like" on Facebook and the like count not changing and the user not being notified until the day after; it simply wouldn't work the same.

For real time to work though, a new way of thinking about the ETL processes is required. Most of what we've talked about so far has assumed that we are processing data in batches, as in taking data that has already been stored somewhere and moving and transforming it. For something to happen in real time though we want to avoid data being at rest as much as possible, as reading and writing from disk increases latency and the point of real time is reducing that latency as much as possible.

Data Streams and Queues

Data streams and queues are the most popular solution to this problem. Streaming allows consistent data "extraction." In reality the data is being extracted from the end of the queue or something similar to a queue. Streaming was an inevitable solution to solve latency problems, as you can take data more frequently, as it's produced. Think of this as quicker and smaller increments that are easier to manage compared to a huge daily bulk load of data. As a bonus, because the data is being ingested as it's being produced, if you can find a way to process it at the same time, you can feedback results to users quicker and also gain faster insights.

There were a few flaws that prevented real-time streaming from being ubiquitous in the past. One major problem was loss of data if there was a fault (i.e., the stream went down so the system couldn't pass the data onto the stream and therefore no data was being collected). Another issue was that real-time tools weren't able to keep up with the velocity. The amount of data that some social networks produce every minute is staggering. Every 60 seconds on Facebook, 510,000 comments are posted, 293,000 statuses are updated, and 136,000 photos are uploaded.

To overcome the problems with streaming, namely reliability and throughput, LinkedIn developed Apache Kafka. Kafka is a publish-subscribe message system that provides isolation between data producers and data consumers. It also allows intermediate storage and buffering of data to help with large velocities. Think of a queueing system at a grocery store. If everyone was to go to a cashier all at the same time and scan their items at the same time, it would be near impossible to unpack it all. To solve this, we have multiple cashiers, and when they're full, we queue working as a buffer (see Figure 5).

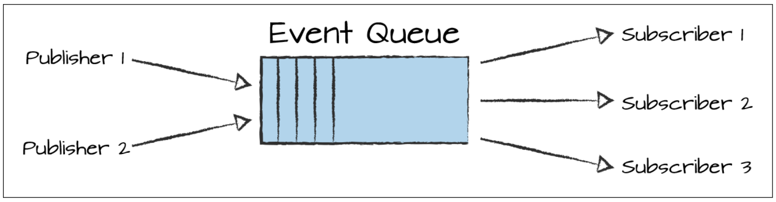

Figure 5. Example of events being added to a queue and consumed by multiple subscribers

The benefit of this is that the system that creates the data is decoupled from the system that reads it, meaning one doesn't directly affect the other. It also means it solves the velocity problem mentioned above, as Kafka buffers data on a queue so the consumer can consume the data at its own pace and therefore doesn't get blocked or miss data.

Again, it follows the same paradigm as the big data storage technologies in terms of scalability. To increase throughput, a data engineer can simply add more machines that read and queue the data, with the idea that you do the same at the processing end so you can also process and take data off the queue as quick as possible, in the same way a supermarket would open more cashier lanes when the current queues become too large.

There are a number of message queue technologies that support real-time data streaming. Some popular examples are Apache Kafka, RabbitMQ, and Amazon's Simple Queue Service (SQS). Each of these services offers its own advantages and has slightly different architectures, all best serving slightly different scenarios. I have focused on Kafka as its story is a great example of what was discussed in the first section of this report: that business growth pushes the boundaries of modern data technologies.

Reliability, Throughput, and Flexibility

It should be fairly obvious by now that when choosing a data streaming platform, there are three main things for a data engineer to consider: reliability, throughput, and flexibility.

Kafka is capable of providing each of these in its own way. To ensure reliability, Kafka uses replication. Each part of the queue is replicated across at least two other replicas, meaning that if one part of the queue goes down, you can simply continue reading from one of the replicas.

Kafka is also fast. LinkedIn benchmarked it and found it can perform two million writes per second on inexpensive hardware. A lot of this speed is due to its low-level interaction with the underlying hardware and is also due to the fact that it can scale sideways, meaning it can write to thousands of partitions spread over many machines.

All of this has allowed companies like Facebook and LinkedIn to produce data at speed, which is quickly shoved onto a queue. For businesses like Facebook and LinkedIn these queues will be horizontally scaled in huge numbers to deal with the throughput and then consumed downstream by a multitude of different systems. Due to the data being queued, you can have multiple systems all reading off the same queue, as data isn't actually removed from a queue when processed and each consumer keeps track of how much of the queue they've read.

So Facebook could have a "Like processor" application that literally just reads Like-related events off the queue, figures out whether it's a Like or an Un-Like, and changes the counter and the list of likes on the post to reflect it, all within milliseconds of the user clicking the Like button.

A large part of data engineering is understanding the end goal and final use case of data and building systems to suit them accordingly. Real-time data is perfect if you have use cases like we have discussed above, where you want things to happen transactionally. If you're just looking to gain business insight, traditional batch processing will probably be fine.

The Backbone of AI and Machine Learning

Earlier in the report, when talking about the transformational aspects of ETL, I gave an example of a crude data science problem and the steps required to solve it. This section will talk about the remainder of the machine learning life cycle and how data engineers have the capability to not only build but productionize a model for use in the real world.

Continuing from the scenario within the transform section, the data engineer has done some ETL and the data is in the data warehouse and ready for use by the data scientists.

The first stage of the machine learning life cycle involves defining the problem space and understanding the expected output. Once this is known, the team can then start prototyping models that can be used to solve this problem.

As the data scientists are building a model, multiple iterations of data ingestion and prep will be cycled through as the data scientists ask more questions and require extra data for answers. The model can be tweaked and tested until the desired output is reached, and due to the collaboration between the scientist and engineer, more time can be spent improving the model's accuracy before it's used in the real world. This prototype stage is its own little cycle of data ingestion and preparation followed by building and testing models.

Once the model is trained to a suitable accuracy, the team breaks out of the mini prototype life cycle and moves onto productionizing the model. If a model is intended to be used regularly—for example, if you are predicting a user's likelihood to purchase from a specific clothing line as we discussed earlier—then the data is going to have to run against the model frequently.

Again, this is where a data engineer can come in handy. They will be able to add the model to a data pipeline that processes your whole user base against the model and segments it accordingly. They can then use the results to trigger automated communications or publish the segments to a data layer to enable segment-targeted content on your website.

At this point there will already be data pipelines to the data warehouse to ingest and transform the data. Adding an extra job of extracting the features from the relevant data sets, passing them through the model, and loading the results into the relevant places isn't going to be very difficult at all.

An example of this machine learning model life cycle is shown in Figure 6, which is taken from Uber's engineering team.

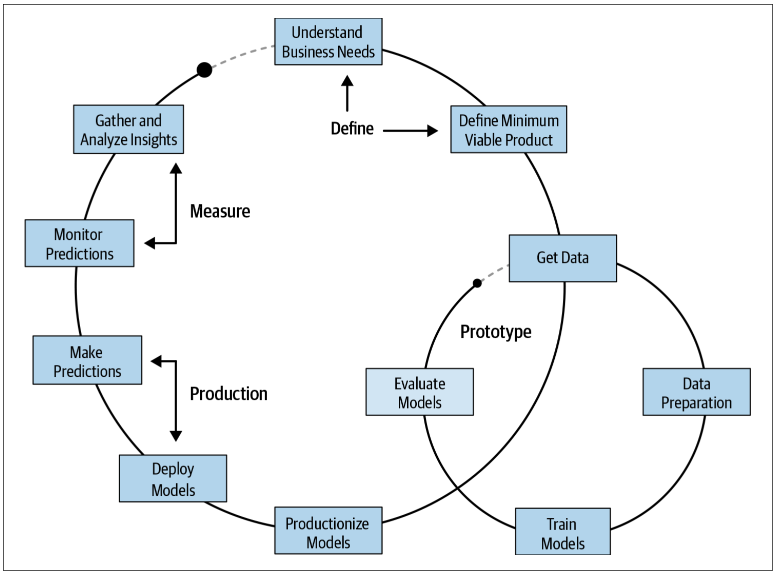

Figure 6. Machine learning model life cycle at Uber

However, if you're concerned about the model becoming stale after a while, you may want to build a continuous feedback loop into the model to help train out any inconsistencies. The engineers, along with the scientists, could turn the code written to build the model into a little service that can be asked to either run some data against the model or rebuild/enhance the model using new data. This again can be engineered into a pipeline with a schedule for rebuilding the data set at relevant times and republishing to production. You should also add some testing steps around this to monitor the performance or accuracy of the new model versus the old one, and if it isn't any better, you can revert back to the old one.

Again, I'm not saying any of this can't be achieved without a data engineer; however, their engineering background and experience building continuously running pipelines lean naturally toward these types of activities.

Optimization

One of the final roles of a data engineer is optimization, especially as data sets have become larger and there has been greater emphasis on real time. This will become more apparent when reading through the Spotify case study in the following section.

The optimization techniques themselves are obviously specific to the technology being used and the final use case. A data engineer isn't always going to know all the techniques either; with there being so many technologies, it's unreasonable to expect them to. However, understanding where to look to optimize based on the use case is a generic skill that every data engineer should have.

Each of the ETL steps can be optimized, from increasing the speed of data extraction to the throughput of a real-time stream. Transformations can be optimized to improve join speeds to other tables and increase the performance of complex calculations or string manipulations. Data loads can be optimized by increasing the speed at which data can be stored in the data store of choice or improving how the data is stored so that it's more effective for the end user.

These optimizations, as I noted previously, are specific to the technology and final use case, so in the next section I will go over all the areas we've discussed so far in this report and align them to a real-world case study as I walk through how Spotify creates their famous hyperpersonalized Discover Weekly playlists and how they deliver it to users every week.

Case Study: Spotify Discover Weekly Playlist

This case study uses information from various presentations that engineers from Spotify delivered in 2015, mainly the "From Idea to Execution: Spotify's Discover Weekly" presentation. They may no longer use some of the techniques or technologies discussed, but the core takeaways from this case study are still relevant.

Spotify is one of the biggest brands in the music industry right now after revolutionizing the way people listen to their favorite music in the modern world. It's also no secret that their key to success is their impressive use of data to keep users coming back to the service by making new music discovery seamless.

One of the key features that users love, and which keeps users coming back to the platform week after week, is their Discover Weekly playlist. The Discover Weekly playlist is a heavily personalized playlist that is curated using data and delivered to every user on the platform each week. This personalized touch and ability to provide new, relevant music to each user on the platform on a weekly basis is unique and is only made possible through solid data engineering backing up amazing data science models. This section will walk through each of the data engineering topics discussed in this report and how they apply to Discover Weekly playlist curation.

Extraction and Real Time

When Spotify published their quarterly statement in fiscal quarter 1 (Q1) of 2019 they had 217 million active users per month (as seen in Figure 7). Each of these users performs searches, listens to songs, skips tracks, curates playlists, and likes songs, all from different areas of the app on a variety of different platforms.

As mentioned in the introduction to this report, this move toward capturing more granular data has been key to the success of the tech giants, because with this data they are able to better understand user behavior and make changes to support what users want. In the case of Spotify, all this log data needs to be captured, and due to the scale of the platform, they have chosen to streamline the process using data streams.

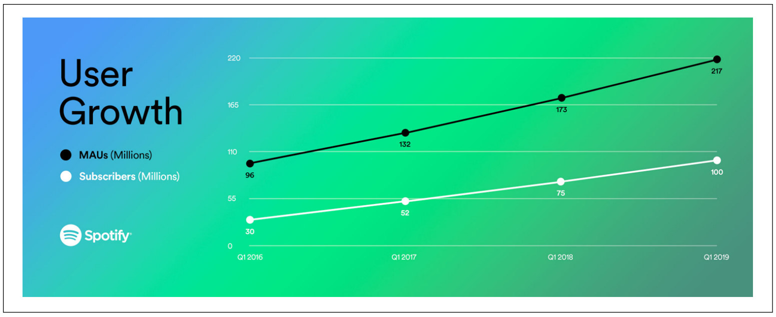

Figure 7. Spotify monthly active users

Log data is fed onto an Apache Kafka queue so it can be processed downstream and extracted into a data lake. This is a consistently open queue that allows the logs to be streamed directly into the data warehouse. This heavily reduces the amount of time between logs being recorded to logs becoming useful. Due to the sheer volume, it also alleviates speed issues with batching this amount of data and copying it over. Even copying batches of this data in hourly intervals for a platform producing data at Spotify's scale would be a mammoth task.

This means that most of the data extraction that takes place is via consumers of the Apache Kafka queues as these queues can directly output data into the data warehouse. On top of the data produced by the platform itself they also have metadata about albums and tracks that are added when artists upload music, and they also extract text data from blogs, news stories, and other online sources—and even song lyrics. These tasks can be performed in isolation from the main Kafka pipeline, and although they don't specifically say how this is done, you can imagine this data is pulled at frequent intervals during the day from various APIs or using web-scraping techniques.

You can see that just managing the logistics of the log data being produced by users of the application is a huge piece of the data engineering in itself and is something that even 10 years ago would have seemed too expensive and practically impossible based on the way data systems were architected at that time.

Now, thanks to the scalability of modern queue platforms such as Kafka, a business producing logs to the scale of Spotify can get access to this data in near–real time, decreasing the time to insight and all at a relatively small price.

Transformation

Now that we know that the raw log data from the platform is being fed into the data warehouse and is streamed via the Kafka queue, we can take a look at how Spotify transforms this into something useful.

The first thing to note is that new data is constantly being added so transformations need to be able to deal with this too. But just because data is continuously arriving doesn't mean you have to process it in real time. In Spotify's case the reason the data is arriving frequently is due to the size; frequent transformations on smaller data sets in this case make sense. This would keep processing times down and again speed up the time from the logs being created to them being transformed into something meaningful and usable downstream.

For Discover Weekly to work, Spotify first needs the logs transformed and data points engineered into features that are put into the model. A simple example is the use of other people's playlists. Imagine you listened to *Stronger* by Kanye West and other songs that fall into that genre (in Spotify's case songs aren't forced into one genre and instead are tagged with multiple labels). Other users who have included that song on playlists themselves in the past week and who also listen to similar genres will likely have put songs into their playlist that you may also like.

Transforming the raw logs to better understand user playlists, listens, and favorite genres all sounds fairly reasonable and something you would expect in order for a feature like this to work; however, Spotify doesn't stop here.

Not only do they use logs of user behavior, but natural language processing (NLP) is also used to transform raw text into something more meaningful and hopefully provide a little more context about what a song or playlist is conveying. As seen in Figure 8, this is performed on news and blog data that is pulled from various sources to help understand how a particular album, song, or artist is described and to again gain more insight and metadata that can be used in building better recommendations.

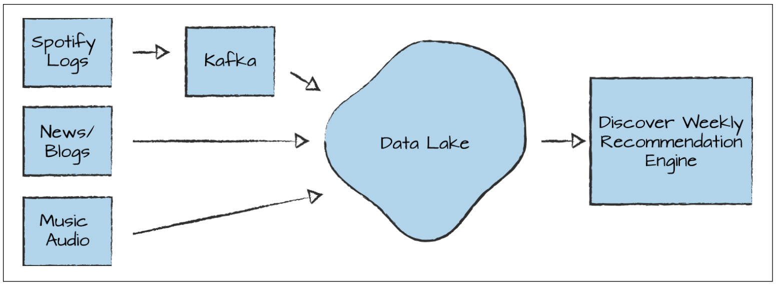

Figure 8. Simplified Discover Weekly data flow

This same technique is even used on playlists. Using the songs within a playlist as words, NLP can be used to try and understand more about the type of playlist based just on the names of the tracks within them. Although I don't know the specifics of Spotify's proprietary process, I can take an informed guess that all of this requires transforming the raw text data, via NLP, into data sets that show frequently used adjectives to describe an artist or album and to even understand the particular "feel" or emotion of a playlist based on track names.

Load

Once the transformation has taken place and the data has made its way through the Discover Weekly recommendation model, they then need to load the playlist for each user back into the platform each week on Sunday night. Let's see how this is done.

First of all, we need to cover where the raw data is loaded once it has been extracted from the Kafka queue. Spotify uses Hadoop as their data lake platform of choice. Hadoop is very flexible in terms of the type of data it can store. We know that the majority of the data is log data extracted from the platform, which can naturally be written to files for processing later. They also process audio files, which wouldn't be suited to traditional database storage so the flexibility of Hadoop allows them to store their audio files alongside the rest of their data. This makes the data easily accessible for use with other data sets.

Hadoop isn't where the final output from the data model is stored though, so let's talk through what the final load process looks like. Loading the final output of Discover Weekly requires refreshing around 75 million playlists on a Sunday, taking time zones into account, so that come Monday morning every user has their own uniquely curated playlist of new songs. This is taking the "load" part of ETL to the next level in terms of extremity and required some clever engineering to make it happen.

Again due to scale, queues were used so that the system could be easily scaled to deal with the throughput. The recommendation system would spit out playlists onto a Rabbit MQ queue (a style of technology similar to Kafka) that would take each playlist, fetch the user's most recent profile picture from Facebook to use as the playlist cover picture, and load them into the database where user playlists are stored. This means the recommendation system can work systematically through users to curate their playlist and add them to the queue, and the queueing system can take care of delivering the playlist to the correct user at the correct time.

As you can see, all the techniques discussed in this report play a huge role in just a single feature of Spotify. Data engineering is used not only to bring data into a data warehouse, but to do it efficiently at scale, where it can be used within machine learning and AI models that are then also engineered into a data pipeline to recommend playlists and deliver them to users consistently, week after week.

These modern data engineering techniques, paired with data science, are what move businesses away from simply understanding what is happening to predicting and dictating what happens next. At Spotify this all happens because of readily scalable architectures that can ingest and quickly materialize raw data into insights of user behavior. This is where most businesses would stop as they now have insight of what their users are doing. What Spotify has successfully done here is take the understanding of what users are doing to help them do more, by providing new but relevant music that keeps them coming back to the platform.

Summary

Whether you're considering hiring a data engineer or are looking at getting into the field of data but don't know where to start, data engineering is clearly an important and interesting field to consider.

Whether it's creating music playlists or powering self-driving cars, data engineering, paired with data science, is one of the most powerful roles within modern businesses, especially those heavily involved with technology.

As data becomes more readily available from disparate sources, consolidating it into one central repository is the only way to get a full 360 degree view of a business. Building this data warehousing platform along with a reporting capability should be step one for any business and obviously relies heavily on data engineers and analysts.

To take the data platform to the next level, the key to success is bringing data scientists alongside data engineers to solve more complex problems, as discussed in the Spotify case study.

If you're undecided about whether data engineer or data scientist is the role for you, hopefully this report gives you a better understanding of the similarities and differences between the two. Data engineers are usually more technical with strong data warehousing and programming backgrounds. Data scientists tend to be more mathematical, but there is a lot of crossover between the roles, notably in programming, as machine learning models usually require writing small applications and heavy data manipulation. It all depends on where your strengths lie and which aspects you enjoy the most.

If you are wondering if data engineers can help solve some of your upcoming business challenges, you should now have a better understanding of just how much of the work involved with pulling business insight or building a machine learning model and making it live are accomplished through data engineering. The biggest and most successful companies in the world rely on this engineering, which allows them to grow, support new features, and most importantly provide users with a product that feels personal. Data engineering should be a key part of any modern business and hopefully this report gives you a glimpse into the possible future of your data platform.

About the Author

Lewis Gavin has been a data engineer for five years working on big data projects for the UK government and transforming the data architecture of a growing charity cashback website. He's also been blogging about skills within the data community for four years on a personal blog.

Lewis studied Computer Science and during his degree worked for the Airbus Helicopter team in Munich enhancing simulator software for military helicopters. He then went on to work for Capgemini who were helping HMRC move into the world of big data. This experience introduced him to data engineering techniques that he has taken into his current role at easyfundraising.org.uk, an online charity cashback site, where he built their data warehousing and reporting capability from the ground up.