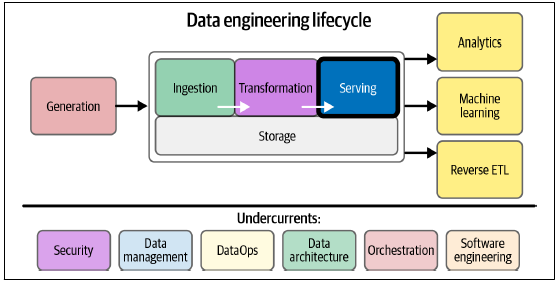
# Fundamentals of Data Engineering - Reis & Housley

## Part II. The Data Engineering Lifecyle in Depth

### Chapter 9 – Serving Data for Analytics, Machine Learning, and Reverse ETL

* We’ve reached the **final stage of the DE lifecycle,** **serving data for downstream use cases**



* We’ll go over various ways to serve data for 3 major use cases you’ll encounter as a DE
* First, you’ll **serve data for analytics and BI**
* You’ll prep data for use in statistical analysis, reporting, + dashboards
* This is the **most traditional area of data serving**
* Arguably, it predates IT and databases, but it is as **important as ever for stakeholders to have visibility into the business, organizational, +financial processes**
* Second, you’ll **serve** **data for ML applications**
* ML is not possible w/out high-quality data that is appropriately prepared
* DE’s work with data scientists and MLE’s to acquire, transform, + deliver the data necessary for model training
* Third, you’ll serve **data through reverse ETL**
* **Reverse ETL**is the process of **sending data back to data sources**
* Ex: Might acquire data from an ad tech platform, run a statistical process on this data to determine cost-per-click bids, + then feed this data back into the ad tech platform
* **Reverse ETL is highly entangled with BI and ML.**
* Before we get into these 3 major ways of serving data, let’s look at some general considerations

#### General Considerations for Serving Data

* Before we get further into serving data, we have a few big considerations
* **First and foremost is trust**, as people need to trust the data you’re providing
* Additionally, you need to understand your **use cases + users**, the **data products** that will be produced, ***how*** you’ll be **serving** data (**self-service** or not), **data definitions and logic**, and **data mesh**
* The **considerations we’ll discuss here are general + apply to any of the 3 ways of serving data**
* **Understanding these considerations will help you be much more effective in serving your data customers**

##### Trust

* **Above all else, trust is the root consideration in serving data, as end users need to trust the data they’re receiving**
* The fanciest, most sophisticated data architecture + serving layer are **irrelevant if end users don’t believe the data is a reliable representation of their business**
* A **loss of trust is often a silent death knell for a data project**, even if the project isn’t officially canceled until months or years later
* **The job of a DE is to serve the best data possible, so you’ll want to make sure your data products always contain high-quality and trustworthy data**
* As you learn to serve data, we’ll reinforce the idea of baking trust into your data + discuss pragmatic ways to accomplish this
* There are too many cases in which data teams are **fixated on pushing out data w/out asking whether stakeholders trust it in the first place**
* Often, stakeholders lose trust in the data
* **Once trust is gone, earning it back is insanely difficult**
* **Inevitably leads to the business not performing to its fullest potential w/ data + data teams losing credibility (and possibly being dissolved)**
* **To realize data quality + build stakeholder trust, utilize data validation + data observability processes, in conjunction w/ visually inspecting + confirming validity with stakeholders**
* **Data validation**is **analyzing data to ensure that it accurately represents financial information, customer interactions, sales, etc.**
* **Data observability** **provides an ongoing view of data and data processes**
* These processes must be applied *throughout the DE lifecycle* to realize a good result as we reach the end
* In addition to building trust in data *quality*, it is **incumbent on DE’s to build trust in their SLAs and SLOs w/ their end users and upstream stakeholders**
* Once users come to depend on data to accomplish business processes, they will require that data is consistently available + up-to-date per the commitments made by DE’s
* **High-quality data is of little value if it’s not available as expected when it’s time to make a critical business decision**
* Note, the SLAs and SLOs may also take the form of **data contracts**(see Chapter 5), formally or informally
* **SLAs come in a variety of forms**
* Regardless of its form, an **SLA tells users what to expect from your data product, as it is a contract between you + your stakeholders.**
* Ex: “Data will be reliably available and of high quality.”
* An **SLO** **is a key part of an SLA + describes the ways you’ll measure performance against what you’ve agreed to**
* Ex: “Our data pipelines to your dashboard or ML workflow will have 99% uptime, with 95% of data free of defects.”
* **Be sure expectations are clear + you have the ability to verify you’re operating w/in your agreed SLA and SLO parameters**
* It’s **not enough to simply agree on an SLA**
* **Ongoing communication is a central feature of a good SLA**
* *Have you communicated possible issues that might affect your SLA or SLO expectations?*
* *What’s your process for remediation and improvement?*
* **Trust is everything. It takes a long time to earn, and it’s easy to lose.**

##### What’s the Use Case and Who’s the User?

* **The serving stage is about data in action**
* But what is a *productive* use of data?
* You need to consider 2 things to answer this question: **what’s the use** **case**, + **who’s the user?**
* The **use case for data goes well beyond viewing reports and dashboards**
* **Data is at its best when it leads to *action(s)***
* *Will an executive make a strategic decision from a report?*
* *Will a user of a mobile food delivery app receive a coupon that entices them to purchase in the next 2 minutes?*
* The **data is often used in more than one use case** (e.g., to train an ML model that does lead scoring *and* populates a CRM (reverse ETL))
* **High-quality, high-impact data will inherently attract many interesting use cases**
* **But in seeking use cases, always ask, “What *action* will this data trigger, and *who* will be performing this action?,” w/ the appropriate follow-up question, “Can this action be *automated*?”**
* **Whenever possible, prioritize use cases w/ the highest possible ROI**
* DE’s love to obsess over the technical implementation details of the systems they build while **ignoring the basic question of *purpose***
* DE’s Engineers want to do what they do best: engineer things
* **When DE’s recognize the need to focus on value + use cases, they become much more valuable and effective in their roles**
* **When starting a new data project, working *backward* is helpful**
* While it’s tempting to focus on tools, try to **start w/ the use case and the users**
* Here are some questions to ask yourself as you get started:
* **Who will use the data, and how will they use it?**
* **What do stakeholders expect?**
* **How can I collaborate w/ data stakeholders (e.g., data scientists, analysts, business users) to understand how the data I’m working w/ will be used?**
* Again, **always approach DE from the perspective of the user + their use case**
* By understanding their expectations + goals, you can work backward to create amazing data products more easily

##### Data Products

* D. J. Patil: “***A good definition of a data product is a product that facilitates an end goal through the use of data***”
* Data products aren’t created in a vacuum
* Like so many other organizational processes we’ve discussed, **making data products is a full-contact sport, involving a mix of product + business alongside tech**
* It’s important to **involve key stakeholders in developing a data product**
* **In *most* companies, a DE is a couple of steps removed from the end users of a data product, so a good DE will seek to fully understand outcomes for direct users such as data analysts + data scientists or customers external to the company**
* When creating a data product, it’s **useful to think of the “jobs to be done.”**
* A user “hires” a product for a “job to be done.”
* This means you **need to know what the user** **wants** (i.e., their **motivation** for “hiring” your product)
* A **classic DE mistake is simply building without understanding the requirements, needs of the end user, or product/market fit**
* This disaster **happens when you build data products nobody wants to use**
* **A good data product has positive feedback loops**
* **More usage of a data product generates more useful data, which is used to improve the data product, then rinse + repeat**
* **When building a data product, keep these considerations in mind:**
* **When someone uses the data product, what do they hope to accomplish?**
* **All too often, data products are made w/out a clear understanding of the outcome expected by the user**
* **Will the data product serve internal or external users?**
* When creating a data product, **knowing whether your customer is internal or external facing will impact the way data is served**
* See Chapter 2 for internal- and external-facing DE
* **What are the outcomes + ROI of the data product you’re building?**
* Building data products that people will use + love is critical
* **Nothing will ruin adoption of a data product more than unwanted utility + loss of trust in the data outputs**
* **Pay attention to the adoption + usage of data products, + be willing to adjust to make users happy**

##### Self-Service or Not?

* ***How* will users interface w/ your data product?**
* Will a business director request a report from the data team, or can this director simply build the report?
* **Self-service data products** **(giving users the ability to build data products on their own) have been a common aspiration of data users for many years**
* What’s better than just giving end users the ability to directly create reports, analyses, + ML models?
* **As of 2022-2023, self-service BI and data science is still mostly aspirational**
* Can occasionally see companies successfully doing self-service w/ data, but this is rare
* Most of the time, attempts at self-service data begin w/ great intentions but ultimately fail, b/c **self-service data is tough to implement in practice**
* Thus, the analyst or data scientist is left to perform the heavy lifting of providing ad hoc reports and maintaining dashboards
* ***Why is self-service data so hard?***
* The answer is nuanced, but it **generally involves understanding the end user**
* If the user is an executive who needs to understand how the business is doing, that person probably just wants a predefined dashboard of clear + actionable metrics
* The executive will likely ignore any self-serve tools for creating custom data views
* If reports provoke further questions, they might have analysts at their disposal to pursue a deeper investigation
* On the other hand, an analyst user might already be pursuing self-service analytics via more powerful tools such as SQL
* **Self-service analytics through a BI layer is not useful**
* The **same considerations apply to data science**
* Although granting self-service ML to “citizen data scientists” has been a goal of many automated ML vendors, adoption is still nascent for the same reasons as self-service analytics
* **In these two extreme cases, a self-service data product is a wrong tool for the job**
* ***Successful* self-service data projects boil down to having the right audience**
* **Identify the self-service users + the “job” they want to do**
* *What are they trying to accomplish by using a self-service data product vs. partnering w/ a data analyst to get the job done?*
* **A group of execs w/ a background in data forms an ideal audience for self-service**, as they likely want to slice + dice data themselves w/out needing to dust off their languishing SQL skills
* **Business leaders willing to invest the time to learn data skills through a company initiative + training program could also realize significant value from self-service**
* **Determine *how* you will provide data to this group**
* What are their **time requirements for new data?**
* What happens if they **inevitably want more data or change the scope of what’s required from self-service?**
* **More data often means more questions, which requires more data**
* You’ll need to **anticipate the growing needs of your self-service users**
* You also need to **understand the fine balance between flexibility + guardrails** **that will help your audience find value + insights w/out incorrect results + confusion**

##### Data Definition and Logic

* As emphatically discussed, the **utility of data in an organization is ultimately derived from its correctness + trustworthiness**
* Critically, **correctness of data goes beyond faithful reproduction of event values from source systems**
* **Data correctness also encompasses proper data definitions and logic**
* These must be baked into data through ALL lifecycle stages, from source systems to data pipelines to BI tools + much more
* **Data definition**refers to **the meaning of data as it is understood throughout the organization**
* Ex: “Customer” has a precise meaning w/in a company + across departments
* When the definition of a customer varies, these must be documented + made available to everyone who uses the data
* **Data logic****stipulates formulas for deriving metrics from data** (say, gross sales or customer lifetime value)
* **Proper data logic must encode data definitions + details of statistical calculations**
* To compute customer churn metrics, we’d need a definition: who is a customer?
* To calculate net profits, we’d need a set of logical rules to determine which expenses to deduct from gross revenue
* **Frequently, data definitions and logic are taken for granted, often passed around the organization in the form of institutional knowledge**
* **Institutional knowledge**takes on a life of its own, often at the expense of **anecdotes replacing data-driven insights, decisions, + actions**
* Instead, ***formally declaring* data definitions + logic both in a data catalog + w/in the systems of the DE lifecycle goes a long way to ensuring data correctness, consistency, + trustworthiness**
* **Data definitions can be served in many ways, sometimes explicitly, but mostly implicitly**
* **“Implicit" means that anytime you serve data for a query, a dashboard, or ML model, the data and derived metrics are presented consistently + correctly**
* When you write a SQL query, you’re implicitly *assuming* the inputs to this query are correct, including upstream pipeline logic + definitions
* **This is where data modeling (Chapter 8) is incredibly useful to capture data definitions and logic in a way that’s understandable + usable by multiple end users**
* Using a **semantic layer**, you **consolidate business definitions + logic in a reusable fashion**
* **Write once, use anywhere**
* This paradigm is an **object-oriented approach to metrics, calculations, and logic**
* *Discussed in more detail later on*

##### Data Mesh

* **Data mesh** (Chapter 3) **will increasingly be a consideration when serving data**, as **it** **fundamentally changes the way data is served within an organization**
* **Instead of siloed data teams** serving their internal constituents, every domain team will have to take on **2 aspects of** **decentralized, peer-to-peer data serving:**
* **1) Teams are responsible for serving data *to other teams* by preparing it for consumption**
* **Data must be good for use in data apps, dashboards, analytics, + BI tools *across* the organization**
* **2) Each team potentially runs its dashboards + analytics for *self-service***
* **Teams consume data from across the organization based on the particular needs in their domain**
* Data consumed from other teams may also make its way into the software designed by a domain team through embedded analytics or an ML feature
* This dramatically changes the details and structure of serving

#### Analytics

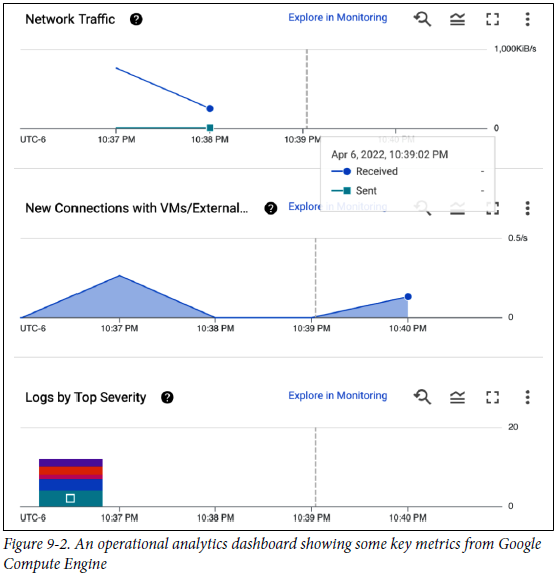
* The **1st data-serving use case you’ll likely encounter is analytics**, which is **discovering, exploring, identifying, + making visible key insights + patterns w/in data**
* Analytics has **many aspects**
* As a practice, analytics is carried out using statistical methods, reporting, BI tools, + more
* **As a DE, knowing the various types + techniques of analytics is key to accomplishing your work**
* This section aims to show how you’ll serve data for analytics + presents some points to think about to help your analysts succeed
* ***Before you even serve data for analytics*, the 1st thing you need to do is identify the end use case**
* *Is the user looking at historical trends?*
* *Should users be immediately + automatically notified of an anomaly, such as a fraud alert?*
* *Is someone consuming a real-time dashboard on a mobile app?*
* These examples highlight the differences between **business analytics** (usually BI), **operational** **analytics**, + **embedded** **analytics**
* Each of these analytics categories has **different goals and unique serving requirements**

##### Business Analytics

* **Business analyticsuses historical + current data to make strategic + actionable decisions**
* The types of decisions tend to **factor in longer-term trends + often involve a mix of statistical + trend analysis, alongside domain expertise + human judgment**
* Business analysis is **as much an art as it is a science**
* Business analytics typically falls into a few big areas like **dashboards, reports, + ad hoc analysis**, + a business analyst might focus on one or all of these categories
* **Understanding an analyst’s workflow will help you, the DE, understand how to serve data**
* A **dashboard****concisely shows decision makers how an organization is performing against a handful of core metrics**, such as sales + customer retention
* These **core metrics are** **presented as visualizations** (e.g., charts or heatmaps), **summary statistics**, or **even a single number**
* Similar to a car dashboard, which gives you a single readout of the critical things you need to know while driving a vehicle
* An organization may have more than one dashboard, w/ **C-level execs using an overarching dashboard + their direct reports using dashboards w/ *their* particular metrics, KPIs, or** **objectives + key results (OKRs)**
* Analysts help create + maintain these dashboards
* Once business stakeholders embrace + rely on a dashboard, the analyst usually responds to requests to look into a potential issue w/ a metric or add a new metric to the dashboard
* Currently, you might use BI platforms to create dashboards, such as Tableau, Looker, Sisense, Power BI, or Apache Superset/Preset
* Analysts are often tasked by business stakeholders w/ creating a **report**
* The **goal of a report is to use data to drive insights + action**
* Ex: An analyst working at an online retail company is asked to investigate which factors are driving a higher-than-expected rate of returns for women’s running shorts
* The analyst runs some SQL queries in the DW, aggregates the return codes that customers provide as the reason for their return, + discovers that the fabric in the running shorts is of inferior quality, often wearing out within a few uses
* Stakeholders such as manufacturing + quality control are notified of these findings
* Furthermore, the **findings are summarized in a report + distributed in the same BI tool where the dashboard resides**
* The analyst was asked to **dig into a potential issue + come back w/ insights**, which represents an example of **ad hoc analysis**
* **Reports typically start as ad hoc requests**
* If the results of an ad hoc analysis are impactful, they often end up in a report or dashboard
* The technologies used for reports + ad hoc analysis are similar to dashboards but may include Excel, Python, R-based notebooks, SQL queries, + much more
* **Good analysts constantly engage w/ the business + dive into the data to answer questions and uncover hidden + counterintuitive trends + insights**
* They **also work w/ DE’s to provide feedback on data quality, reliability issues, + requests for new datasets**
* The **DE is responsible for addressing this feedback + providing new datasets for the analyst to use**
* Returning to the running shorts example, suppose that after communicating their findings, analysts learn that manufacturing can provide them w/ various supply chain details regarding the materials used in the running shorts
* DE’s undertake a project to ingest this data into the DW
* Once the supply chain data is present, analysts can correlate specific garment serial numbers w/ the supplier of the fabric used in the item
* They discover that most failures are tied to 1 of their 3 suppliers, + the factory stops using fabric from this supplier
* **The data for business analytics is frequently served in *batch* mode from a DW or a data lake**
* This **varies wildly across companies, departments, + even data teams w/in companies**
* New data might be available every second, every minute, every 30 minutes, every day, or once a week
* The **frequency of the batches can vary for several reasons**
* One key thing to note is that **DE’s working on analytics problems should consider various potential applications of data (current + future)**
* **It is common to have mixed data update frequencies to serve use cases appropriately, but remember that the frequency of ingestion sets a ceiling on downstream frequency**
* **If streaming applications exist for the data, it should be ingested as a stream even if some downstream processing + serving steps are handled in batches**
* Of course, **DE’s must address various backend technical considerations in serving business analytics**
* **Some BI tools store data in an internal storage layer**
* **Other tools run queries on your data lake or DW**
* This is **advantageous b/c you can take full advantage of your OLAP database’s power**
* But as discussed, the **downside is cost, access control, + latency**

##### Operational Analytics

* If **business analytics is about using data to discover actionable insights**, then **operational analytics uses data to take *immediate* action**
* **Operational analytics vs. business analytics = immediate action vs. actionable insights**
* **The big difference between operational v. business analytics is *time***
* Data used in **business analytics takes a longer view** of the question under consideration
* Up-to-the-second updates are nice to know but won’t materially impact the quality or outcome
* **Operational analytics** is quite the opposite, as **real-time updates can be impactful in addressing a problem when it occurs**
* An example of operational analytics is real-time application monitoring
* Many SWE teams want to know how their application is performing, + if issues do arise, they want to be notified immediately
* The engineering team might have a dashboard that shows key metrics such as requests per second, database I/O, or whatever metrics are important



* Certain conditions can trigger scaling events, adding more capacity if servers are overloaded
* If certain thresholds are breached, the monitoring system might also send alerts via text message, group chat, + email
* Let’s return once again to our running shorts example
* Using analytics to discover bad fabric in the supply chain was a huge success, + now business leaders + DE’s want to find *more* opportunities to utilize data to improve product quality
* The DE’s suggest deploying real-time analytics at the factory
* The plant already uses a variety of machines capable of streaming real-time data
* In addition, the plant has cameras recording video on the manufacturing line
* Right now, technicians watch the footage in real time, look for defective items, + alert those running the line when they see a high rate of snags appearing in items
* DE’s realize that they can use an off-the-shelf cloud machine vision tool to identify defects in real time automatically
* Defect data is tied to specific item serial numbers + then streamed
* From here, a real-time analytics process can tie defective items to streaming events from machines further up the assembly line
* Using this approach, factory floor analysts discover that the quality of raw fabric stock varies significantly from box to box
* When the monitoring system shows a high rate of snag defects, line workers can remove the defective box + charge it back to the supplier
* Seeing the success of this quality improvement project, the supplier decides to adopt similar quality-control processes
* DE’s from the retailer work w/ the supplier to deploy their real-time data analytics, dramatically improving the quality of their fabric stock

###### Business and Operational Analytics

* **The line between business and operational analytics has begun to blur**
* As streaming + low-latency data become more pervasive, it is only natural to apply operational approaches to business analytics problems
* In addition to monitoring website performance on Black Friday, an online retailer could also analyze and present sales, revenue, + the impact of advertising campaigns in real time
* The **data architectures will change to fit into a world where you can have both your red hot and warm data in one place**
* The central question you should always ask yourself, + your stakeholders, is this: **if you have streaming data, *what are you going to do with it?*** ***What action should you take?***
* **Correct action creates impact + value**
* **Real-time data w/out action is an unrelenting distraction**
* **In the long term**, one may predict that **streaming may supplant batch**
* Data products over the next 10 years will likely be streaming-first, w/ the ability to seamlessly blend historical data
* After real-time collection, data can still be consumed + processed in batches as required

##### Embedded Analytics

* Whereas **business + operational analytics are *internally* focused**, a recent trend is ***external*-facing or embedded analytics**
* W/ so much data powering applications, companies increasingly provide analytics to end users
* These are typically referred to as **data applications**, **often w/ analytics dashboards embedded w/in the application itself**
* Also known as **embedded analytics**, these **end-user-facing dashboards give users key metrics about their relationship with the application**
* Ex: A smart thermostat has a mobile app that shows temperature in real time + up-to-date power consumption metrics, allowing the user to create a better energy-efficient heating or cooling schedule
* Ex: A third-party ecommerce platform provides its sellers a real-time dashboard on sales, inventory, + returns
* The seller has the option to use this information to offer deals to customers in near real time
* In both cases, **an application allows users to make real-time decisions (manually or automatically) based on data**
* The landscape of embedded analytics is snowballing, + expect that such data applications will become increasingly pervasive within the next few years
* As a DE, you’re probably not creating the embedded analytics frontend, as the application developers handle that
* **Since you’re responsible for the databases *serving* the embedded analytics, you’ll need to understand the speed + latency requirements for embedded analytics**
* **Performance for embedded analytics encompasses 3 problems**
* **1) App users are not as tolerant of infrequent batch processing as internal company analysts**
* Users of a recruiting SaaS platform may expect to see a change in their statistics as soon as they upload a new resume
* **Users want low data latency**
* **2) Users of data apps expect fast query performance**
* When they adjust parameters in an analytics dashboard, they want to see refreshed results appear in seconds
* **3) Data apps must often support extremely high query rates across many dashboards and numerous customers**
* **High concurrencyis critical**
* Google + other early major players in the data apps space developed exotic technologies to cope with these challenges
* For new startups, the default is to use conventional transactional databases for data applications
* As their customer bases expand, they outgrow their initial architecture
* They have access to a **new generation of databases that combine high performance (fast queries, high concurrency, + near real-time updates) w/ relative ease of use (e.g., SQL-based analytics)**

#### Machine Learning

* The second major area for serving data is machine learning (ML), which is increasingly common
* W/ the rise of MLE (itself almost a parallel universe to DE), you might ask yourself where a DE fits into the picture
* Admittedly, the boundary between ML, data science, DE, + MLE is increasingly fuzzy, + this boundary varies dramatically between organizations
* In some organizations, MLE’s take over data processing for ML applications right after data collection or may even form an entirely separate + parallel data organization that handles the entire lifecycle for all ML applications
* DE’s handle all data processing in other settings + then hand off data to MLE’s for model training
* DE’s may even handle some extremely ML-specific tasks, such as featurization of data
* Let’s return to our example of quality for control of running shorts produced by an online retailer
* Suppose that streaming data has been implemented in the factory that makes the raw fabric stock for the shorts
* Data scientists discovered that the quality of the manufactured fabric is susceptible to characteristics of the input raw polyester, temperature, humidity, + various tunable parameters of the loom that weaves the fabric
* They then develop a basic model to optimize loom parameters
* MLE’s automate model training + set up a process to automatically tune the loom based on input parameters
* Data and ML engineers work *together* to design a featurization pipeline, + DE’s implement and maintain the pipeline

#### What a Data Engineer Should Know About Machine Learning

* Before we discuss serving data for ML, you may ask yourself how much ML you need to know as a DE
* ML is an *incredibly* vast topic, + countless books + courses are available to learn ML.
* **While a DE doesn’t need to have a *deep* understanding of ML, it helps tremendously to know the basics of how classical ML works + the fundamentals of DL**
* **Knowing the basics of ML will go a long way in helping work alongside data scientists in building data products**
* Here are **some areas of ML that a DE should be familiar with**:
* The difference between **supervised, unsupervised, + semi-supervised learning**
* The difference between **classification and regression techniques**
* The various techniques for **handling time-series data**
* This includes time-series **analysis**, as well as time-series **forecasting**
* When to use the **“classical” techniques** (logistic regression, tree-based learning, support vector machines) **versus DL**
* Can constantly see data scientists immediately jump to DL when it’s overkill
* As a DE, your basic knowledge of ML can help you spot whether an ML technique is appropriate + scales the data you’ll need to provide.
* When would you use **automated machine learning (AutoML) versus handcrafting** an ML model?
* What are the trade-offs with each approach regarding the data being used?
* What are **data-wrangling techniques** used for **structured and unstructured data?**
* All data that is used for ML is converted to numbers
* If you’re serving structured or semi-structured data, ensure that the data can be properly converted during the feature-engineering process
* How to **encode categorical data** and the **embeddings for various types of data**
* The difference between **batch and online learning**
* Which approach is appropriate for your use case?
* **How does the DE lifecycle intersect with the ML lifecycle at your company?**
* Will you be responsible for interfacing w/ or supporting ML technologies such as **feature stores** or **ML observability?**
* Know when it’s appropriate to **train locally, on a cluster, or at the edge**
* When would you use a **GPU over a CPU?**
* The type of hardware you use largely depends on the type of ML problem you’re solving, the technique you’re using, *and* the size of your dataset.
* Know the difference between the applications of **batch and streaming data** **in training ML models**
* Ex: Batch data often fits well w/ offline model training, while streaming data works with online training
* What are **data cascades**, and how might they **impact ML models?**
* Are **results returned in real time or in batch?**
* Ex: A batch speech transcription model might process speech samples + return text in batch after an API call
* Ex: A product recommendation model might need to operate in real time as a customer interacts w/ an online retail site
* The use of **structured versus unstructured data**
* We might cluster tabular (structured) customer data or recognize images (unstructured) by using a neural net
* Again, ML is a *vast* subject area, and this book won’t teach you these topics, or even ML generalities
* If you’d like to learn more about ML, try reading *Hands on Machine Learning with Scikit-Learn, Keras, and TensorFlow* by Aurelien Geron (O’Reilly)
* <https://www.oreilly.com/library/view/hands-on-machine-learning/9781492032632/>
* Countless other ML courses and books are available online
* Because books + online courses evolve so rapidly, do research on what seems like a good fit for you

#### Ways to Serve Data for Analytics and Machine Learning

* As with analytics, **DE’s provide data scientists + MLE’s w/ the data they need to do their jobs**
* We have placed serving for ML alongside analytics b/c the pipelines and processes are extremely similar
* There are many ways to serve data for analytics + ML
* Some common ways to serve this data include files, databases, query engines, + data sharing

##### File Exchange

* **File exchange is ubiquitous in data serving**
* We **process data + generate files to pass to data consumers**
* Keep in mind that **a file might be used for many purposes**
* Ex: A data scientist might load a text file (unstructured data) of customer messages to analyze the sentiments of customer complaints
* Ex: A business unit might receive invoice data from a partner company as a collection of CSVs (structured data), + an analyst must perform some statistical analysis on these files
* Ex: A data vendor might provide an online retailer w/ images of products on a competitor’s website (unstructured data) for automated classification using computer vision
* **The way you serve files depends on several factors, such as**:
* **Use case** (business analytics, operational analytics, embedded analytics)
* The **data consumer’s data-handling processes**
* This is **one of the main considerations**
* It is often necessary to serve data through files rather than data sharing b/c the data consumer cannot use a sharing platform
* The **size and number of individual files in storage**
* **Who is accessing** this file
* **Data type** (structured, semi-structured, or unstructured)
* The simplest file to serve is something along the lines of emailing a single Excel file
* This is still a common workflow even in an era when files can be collaboratively shared
* The problem w/ emailing files is each recipient gets *their version* of the file.
* If a recipient edits the file, these edits are specific to that user’s file
* Deviations among files inevitably result
* And what happens if you no longer want the recipient to have access to the file?
* If the file is emailed, you have very little recourse to retrieve the file.
* If you *need* a coherent, consistent version of a file, try using a collaboration platform such as Microsoft 365 or Google Docs
* **Of course, serving single files is hard to scale, + your needs will eventually outgrow simple cloud file storage**
* You’ll likely grow into an **object storage bucket** if you have a **handful of large files**, or a **data lake** if you have a **steady supply of files**
* **Object storage can store any type of blob file + is especially useful for semi-structured or unstructured files**
* People generally consider file exchange through object storage (data lake) to land under “data sharing” rather than file exchange since the process can be significantly more scalable and streamlined than ad hoc file exchange

##### Databases

* Databases are a critical layer in serving data for analytics + ML
* For this discussion, we’ll implicitly keep our focus on serving data from OLAP databases (e.g., DW’s and data lakes)
* **Serving data involves querying a database and then *consuming* those results for a use case**
* An analyst or data scientist might query a database by using a SQL editor + export those results to a CSV file for consumption by a downstream application, or analyze the results in a notebook
* **Serving data from a database carries a variety of benefits**
* **Imposes order and structure on the data through schema**
* **Can offer fine-grained permission controls at the table, column, + row level**, allowing DBA’s to craft complex **access policies for various roles**
* **Can offer high serving performance for large, computationally intensive queries + high query concurrency**
* BI systems usually share the data processing workload w/ a source database, but the boundary between processing in the two systems varies
* Ex: A Tableau server runs an initial query to pull data from a database + stores it locally
* Basic OLAP/BI slicing + dicing (interactive filtering + aggregation) runs directly on the server from the local data copy
* Ex: On the other hand, Looker (+ similar modern BI systems) relies on a computational model called **query pushdown**
* Looker encodes data processing logic in a specialized language (LookML), combines this w/ dynamic user input to generate SQL queries, runs these against the source database, + presents the output
* Both Tableau + Looker have **various configuration options for caching results to reduce the processing burden for frequently run queries**
* A data scientist might connect to a database, extract data, + perform feature engineering + selection
* This converted dataset is then fed into an ML model, + the offline model is trained + produces predictive results.
* **DE’s are quite often tasked w/ managing the database-serving layer**
* This **includes management of performance + costs**
* In databases that separate compute + storage, this is a somewhat more subtle optimization problem than in the days of fixed on-prem infrastructure
* Ex: It is now possible to spin up a new Spark cluster or Snowflake warehouse for each analytical or ML workload
* It is **generally recommended to at least split out clusters by major use cases, such as ETL and serving for analytics + data science**
* Often data teams choose to slice more finely, assigning one warehouse per major area
* This makes it possible for different teams to budget for their query costs under the supervision of a DE team
* Recall **3 performance considerations** from earlier: **data latency, query performance, concurrency**
* A system that can ingest *directly* from a stream can lower data latency
* And many database architectures rely on SSD or memory caching to enhance query performance + concurrency to serve the challenging use cases inherent in embedded analytics
* Increasingly, data platforms like Snowflake + Databricks allow analysts + data scientists to operate under a single environment, providing SQL editors + data science notebooks under one roof
* B/c compute + storage are separated, the analysts + data scientists can consume the underlying data in various ways w/out interfering with each other
* This will allow high throughput and faster delivery of data products to stakeholders

##### Streaming Systems

* Streaming analytics are increasingly important in the realm of serving
* At a high level, understand that this type of serving may involve **emitted metrics**, which are different from traditional queries
* Also, we see **operational analytics databases** (discussed earlier) playing a growing role in this area
* These databases **allow queries to run across a large range of historical data, encompassing up-to-the-second current data**
* Essentially, they **combine aspects of OLAP databases w/ stream-processing systems**
* **Increasingly, you’ll work w/ streaming systems to serve data for analytics + ML, so get familiar with this paradigm**
* For an idea of where streaming systems are going, see the **live data stack** in Chapter 11

##### Query Federation

* Recall **query federation pulls data from multiple sources, such as data lakes, RDBMSs, + DW’s**
* **Federation is becoming more popular as distributed query virtualization engines gain recognition as ways to serve queries w/out going through the trouble of centralizing data in an OLAP system**
* Today, you can find OSS options like Trino and Presto, + managed services such as Starburst
* Some of these offerings describe themselves as ways to enable the data mesh, but time will tell how that unfolds.
* **When serving data for federated queries, be aware that the end user might be querying several systems (OLTP, OLAP, APIs, filesystems, etc.)**
* Instead of serving data from a single system, you’re **now serving data from multiple systems, each w/ its own usage patterns, quirks**, + nuances
* This **poses challenges for serving data**.
* If federated queries touch live production source systems, you *must* ensure that the federated query won’t consume excessive resources in the source
* **Typically, federated queries are ideally suited when you want flexibility in analyzing data or the source data needs to be tightly controlled**
* Federation **allows ad hoc queries for performing exploratory analysis, blending data from various systems w/out the complexity of setting up data pipelines or ETL**
* This will **allow you to determine whether the performance of a federated query is sufficient for ongoing purposes or if you need to set up ingestion on some or all data sources + centralize the data in an OLAP database or data lake**
* Federated queries **also provide read-only access to source systems**, which is **great when you don’t want to serve files, database access, or data dumps**
* The end user reads only the version of the data they’re supposed to access + nothing more
* **Query federation is a great option to explore for situations where access + compliance are critical**

##### Data Sharing

* See Chapter 5 for an extensive discussion of data sharing
* **Any data exchange between organizations or units w/in a larger organization can be viewed as data sharing**
* Still, it **typically often means specifically sharing through massively multitenant storage systems in a cloud environment**
* **Data sharing generally turns data serving into a security and access control problem**
* The **actual queries are now handled by the data consumers (analysts and data scientists) rather than the engineers sourcing the data**
* Whether serving data in a data mesh w/in an organization, providing data to the public, or serving to partner businesses, data sharing is a compelling serving model
* Data sharing is **increasingly a core feature of major data platforms** like Snowflake, Redshift, and BigQuery, **allowing companies to share data safely + securely w/ each other**

##### Semantic and Metrics Layers

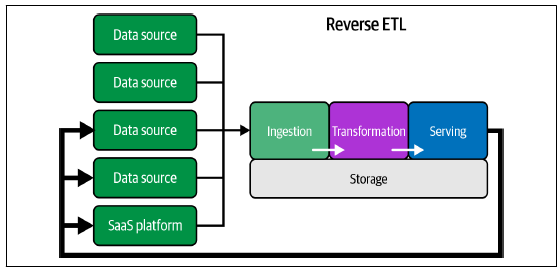
* **When DE’s think about “serving”, they naturally tend to gravitate toward the data processing and storage technologies** (i.e., will you use Spark or a cloud DW? Is your data stored in object storage or cached in a fleet of SSDs?)
* **But powerful processing engines that deliver quick query results across vast datasets don’t inherently make for quality business analytics**
* **When fed poor-quality data or poor-quality queries, powerful query engines quickly return bad results**
* **Whereas data quality focuses on characteristics of the data itself + various techniques to filter or improve bad data, *query* quality is a question of building a query w/ appropriate logic that returns accurate answers to business questions**
* **Writing high-quality ETL queries + reporting is time-intensive, detailed work**
* Various tools can help automate this process while facilitating consistency, maintenance, and continuous improvement.
* Fundamentally, a **metrics layer****is a tool for maintaining + computing business logic**
* Benn Stancil, “*The Missing Piece of the Modern Data Stack*” April 22, 2021
* <https://oreil.ly/wQyPb>
* A **semantic layer**is extremely similar conceptually, + **headless BI**is another closely related term
* Srini Kadamati, “*Understanding the Superset Semantic Layer*” December 21, 2021,
* <https://oreil.ly/6smWC>
* This **metrics layer can live in a BI tool or in software that builds transformation queries**
* 2 concrete examples are Looker and Data Build Tool (dbt)
* Looker’s LookML allows users to define virtual, complex business logic
* Reports + dashboards point to specific LookML for computing metrics
* Looker allows users to define standard metrics + reference them in many downstream queries
* This is meant to solve the traditional problem of repetition + inconsistency in traditional ETL scripts
* Looker uses LookML to generate SQL queries, which are pushed down to the database
* Results can be persisted in the Looker server or in the database itself for large result sets
* **dbt allows users to define complex SQL data flows encompassing many queries + standard definitions of business metrics**, much like Looker
* Unlike Looker, **dbt runs exclusively in the transform layer, although this can include pushing queries into views that are computed at query time**
* Whereas Looker focuses on serving queries + reporting, **dbt can serve as a robust data pipeline orchestration tool for analytics engineers**
* **The metrics layer tools will likely grow more popular w/ wider adoption + more entrants, as well as move upstream toward the application**
* Metrics layer tools **help solve a central question in analytics that has plagued organizations since people have analyzed data: “*Are these numbers correct?*”**
* Many new entrants are in the space beside the ones mentioned above

##### Serving Data in Notebooks

* Data scientists often use notebooks in their day-to-day work
* Whether it’s exploring data, engineering features, or training a model, the data scientist will likely use a notebook
* As of 2022-2023, the most popular notebook platform is Jupyter Notebook, along w/ its next-generation iteration, JupyterLab
* Jupyter is open source + can be hosted locally on a laptop, on a server, or through various cloud-managed services
* “Jupyter” stands for “Julia, Python, and R*”,* the latter two being popular for data science applications, especially notebooks
* Regardless of the language used, **the first thing you’ll need to consider is how data can be accessed from a notebook**
* Data scientists will ***programmatically* connect to a data source**, such as an API, database, DW, or data lake (i.e., a notebook can be served data from many sources, such as object storage or a database, DW, or data lake)
* In a notebook, **all connections are created using the appropriate built-in or imported libraries** to load a file from a filepath, connect to an API endpoint, or make an ODBC connection to a database
* A **remote connection may require the correct credentials + privileges to establish a connection**
* **Incorrectly handled credentials in notebooks + data science code are a major security risk**
* It is common to embed credentials *directly* in code, where they often leak into version control repos
* Credentials are also frequently passed around through messages and email.
* **DE’s should audit data science security practices + work collaboratively on improvements**
* Data scientists are highly receptive to these conversations if given alternatives
* **DE’s should set standards for handling credentials**
* **Credentials should never be embedded in code**
* **Ideally, data scientists use credential managers or CLI tools to manage access.**
* Once connected, a user may need the correct access to tables (and rows/columns) or files stored in object storage
* The DE will often assist the data scientist in finding the right data + then ensure that they have the right permissions to access the rows + columns required
* One incredibly common workflow for data scientists is running a local notebook and loading data into a pandas dataframe
* Pandasis a prevalent Python library used for data manipulation + analysis and is commonly used to load data (say, a CSV file) into a Jupyter notebook
* **When pandas loads a dataset, it stores this dataset *in memory***
* *What happens when the dataset size exceeds the local machine’s available memory?* (inevitably happens given the limited memory of laptops + workstations) It stops a data science project dead in its tracks
* It’s time to **consider more scalable options**
* First, **move to a cloud-based notebook where the underlying storage + memory for the notebook can be flexibly scaled**
* Upon outgrowing this option, look at **distributed execution systems** (popular Python-based options include Dask, Ray, and Spark)
* **If a full-fledged cloud-managed offering seems appealing, consider setting up a data science workflow** using Amazon SageMaker, Google Cloud Vertex AI, or Microsoft Azure Machine Learning
* **Finally, open source end-to-end ML workflow** options such as Kubeflow and MLflow **make it easy to scale ML workloads** in Kubernetes and Spark, respectively
* **The point is to get data scientists off their laptops + take advantage of the cloud’s power and scalability**
* **DE’s and MLE’s play a key role in facilitating the move to scalable cloud infrastructure**
* The exact division of labor depends a great deal on the details of your organization
* **The two roles should take the lead in setting up cloud infrastructure, overseeing the management of environments, + training data scientists on cloud-based tools**
* **Cloud environments require significant operational work, such as managing versions + updates, controlling access, + maintaining SLAs**
* **As w/ other operational work, a significant payoff can result when “data science ops” are done well**
* Notebooks may even become a part of production data science (they are widely deployed at Netflix)
* This is an interesting approach with **advantages and trade-offs**
* **Productionized notebooks allow data scientists to get their work into production much faster, but they are also inherently a substandard form of production**
* The **alternative is to have MLE’s + DE’s convert notebooks for production use, placing a significant burden on these teams**
* **A hybrid of these approaches may be ideal, w/ notebooks used for “light” production and a full production-ization process for high-value projects**

#### Reverse ETL

* Today, **reverse ETL**is a buzzword that **describes serving data by loading it from an OLAP database back into a source system**
* That said, any DE who’s worked in the field for more than a few years has probably done some variation of reverse ETL
* Reverse ETL grew in popularity in the late 2010s/early 2020s + is increasingly recognized as a formal DE responsibility.
* A DE might pull customers + order data from a CRM + store it in a DW
* This data is used to train a lead scoring model, whose results are returned to the DW
* Your company’s sales team wants access to these scored leads to try to generate more sales
* You have a few options to get the results of this lead scoring model into the hands of the sales team
* You can put the results in a dashboard for them to view
* Or might email the results to them as an Excel file
* The challenge with these approaches is that *they are not connected to the CRM*, where a salesperson does their work
* *Why not just put the scored leads back into the CRM?*
* As mentioned, **successful data products reduce friction with the end user**
* In this case, the end user is the sales team.
* Using reverse ETL + loading the scored leads back into the CRM is the easiest + best approach for this data product
* **Reverse ETL takes processed data from the *output* side of the DE lifecycle + feeds it back into source systems**



* *How do you begin serving data with reverse ETL?*
* While you can roll your reverse ETL solution, many off-the-shelf reverse ETL options are available
* Try using OSS, or a commercial managed service
* That said, **the reverse ETL space is changing extremely quickly**
* No clear winners have emerged, + many reverse ETL products will be absorbed by major clouds or other data product vendors, **choose carefully**
* A **few words of warning regarding reverse ETL:**
* **Reverse ETL inherently creates feedback loops**
* Ex: Imagine we download Google Ads data, use a model to compute new bids, load the bids back into Google Ads, + start the process again
* Suppose that because of an error in your bid model, the bids trend ever higher, + your ads get more and more clicks
* You can quickly waste massive amounts of money!
* **Be careful, and build in monitoring + guardrails**

#### Whom You’ll Work With

* As discussed, in the serving stage, a DE will interface with a lot of stakeholders who include (but aren’t limited to) the following:
* Data analysts
* Data scientists
* MLOps/ML engineers
* The business (non-data or non-technical stakeholders, managers, + executives)
* As a reminder, **the DE operates in a *support* role for these stakeholders + is not necessarily responsible for the end uses of data**
* Ex: A DE supplies the data for a report that analysts interpret, but the DE isn’t responsible for these interpretations
* Instead, **the DE is responsible for producing the highest-quality data products possible**
* **A DE should be aware of feedback loops between the DE lifecycle + the broader use of data once it’s in the hands of stakeholders**
* **Data is rarely static, + the outside world will influence the data that is ingested + served + re-ingested + re-served**
* **A big consideration for DE’s in the serving stage of the lifecycle is the separation of duties and concerns**
* If at an early-stage company, the DE may also be an MLE or data scientist, which is not sustainable
* As the company grows, you **need to establish a clear division of duties w/ other data team members**
* **Adopting a data mesh dramatically reorganizes team responsibilities, + every domain team takes on aspects of serving**
* **For a data mesh to be successful, each team must work effectively on its data-serving responsibilities, + teams must also effectively collaborate to ensure organizational success**

#### Undercurrents

* The undercurrents come to finality with serving
* Remember that the DE lifecycle is just that, a lifecycle (What goes around comes around)
* **One could see many instances where serving data highlights something missed earlier in the lifecycle**
* **Always be on the lookout for how the undercurrents can help you spot ways to improve data products**
* “Data is a silent killer,” + the undercurrents come to a head in the serving stage
* **Serving is your final chance to make sure your data is in great shape before it gets into the hands of end users**

##### Security

* **The same security principles apply whether sharing data with people or systems**
* One can often see data shared indiscriminately, w/ little to no access controls or thought as to what the data will be used for
* This is a huge mistake that can have catastrophic results, such as a data breach + the resulting fines, bad press, + lost jobs
* **Take security seriously, especially in this stage of the lifecycle**
* **Of all the lifecycle stages, serving presents the largest security surface**
* **As always, exercise the *principle of least privilege* both for people and systems, + provide only the access required for the purpose at hand and the job to be done**
* What data does an executive need vs. an analyst or data scientist?
* What about an ML pipeline or reverse ETL process?
* **These users + destinations all have different data needs, + access should be provided accordingly**
* **Avoid giving carte blanche permissions to everyone and everything**
* **Serving data is often read-only unless a person or process needs to update data in the system from which it is queried**
* **People should be given read-only access to specific databases + datasets unless their role requires something more advanced like write or update access**
* This can be accomplished by combining groups of users w/ certain IAM roles (i.e., analysts group, data scientist group) or custom IAM roles if this makes sense
* **For systems, provide service accounts and roles in a similar fashion**
* **For both users *and* systems, narrow access to a dataset’s fields, rows, columns, + cells if this is warranted**
* **Access controls should be as fine-grained as possible + revoked when access is no longer required**
* Access controls are **critical when serving data in a multitenant environment**
* **Make sure users can access only *their* data + nothing more**
* A good approach is to **mediate access through filtered views**, thus alleviating the security risks inherent in sharing access to a common table
* Another suggestion is to **use data sharing in your workflows**, which allows for read-only granular controls between you + people consuming your data.
* **Check how often data products are used + whether it makes sense to stop sharing certain data products**
* It’s extremely common for an executive to urgently request an analyst to create a report, only to have this report very quickly go unused
* **If data products aren’t used, ask the users if they’re still needed**
* If not, kill off the data product
* This means one less security vulnerability floating around.
* Finally, **view access control and security not as impediments to serving but as key enablers**
* There have been many instances where complex, advanced data systems were built, potentially having a significant impact on a company
* And b/c security was not implemented correctly, few people were allowed to access the data, so it languished
* **Fine-grained, robust access control means that more interesting data analytics + ML can be done while still protecting the business + its customers**

##### Data Management

* You’ve been incorporating data management along the DE lifecycle, + the impact of your efforts will soon become apparent as people use your data products
* **At the serving stage, you’re mainly concerned w/ ensuring that people can access high-quality and trustworthy data**
* As we mentioned, **trust is perhaps the most critical variable in data serving**
* **If people trust their data, they will use it, + untrusted data will go unused**
* Be sure to **make data trust and data improvement an active process by providing feedback loops**
* **As users interact w/ data, they can report problems + request improvements**
* **Actively communicate back to your users as changes are made**
* *What data do people need to do their jobs?*
* Especially w/ regulatory + compliance concerns weighing on data teams, **giving people access to the raw data (even w/ limited fields + rows) poses a problem of tracing data back to an entity, such as a person or a group of people**
* Thankfully, advancements in data obfuscation allow you to serve synthetic, scrambled, or anonymized data to end users
* These “fake” datasets should sufficiently allow an analyst or data scientist to get the necessary signal from the data, but in a way that makes identifying protected information difficult
* Though this isn’t a perfect process (w/ enough effort, many datasets can be de-anonymized or reverse-engineered), it at least reduces the risk of data leakage
* Also, **incorporate semantic and metrics layers into your serving layer, alongside rigorous data modeling that properly expresses business logic and definitions**
* This **provides a single source of truth, whether for analytics, ML, reverse ETL, or other serving uses**

##### DataOps

* **The steps you take in data management (data quality, governance, + security) are *monitored* in DataOps** (Essentially, **DataOps *operationalizes* data management**)
* The following are **some things to monitor**:
* Data health + data downtime
* Latency of systems serving data (dashboards, databases, etc.)
* Data quality
* Data + system security and access
* Data + model versions being served
* Uptime to achieve an SLO
* **A variety of new tools have sprung up to address various monitoring aspects**
* Ex: Many popular data observability tools aim to minimize *data downtime +* maximize data quality
* Observability tools may cross over from data to ML, supporting monitoring of models + model performance
* **More conventional DevOps monitoring is also critical to DataOps (e.g., you need to monitor whether connections are stable among storage, transformation, and serving)**
* **As in every stage of the DE lifecycle, version-control code + operationalize deployment**
* This applies to analytical code, data logic code, ML scripts, + orchestration jobs
* **Use multiple stages of deployment (dev, test, prod) for reports + models**

##### Data Architecture

* **Serving data should have the same architectural considerations as other DE lifecycle stages**
* At the serving stage, **feedback loops must be fast and tight**
* **Users should be able to access the data they need as quickly as possible when they need it**
* Data scientists are notorious for doing most development on their local machines.
* As discussed earlier, **encourage them to migrate these workflows to common systems in a cloud environment, where data teams can collaborate in DEV, TEST, and PROD environments and create proper production architectures**
* **Facilitate analysts + data scientists by supporting tools for publishing data insights w/ little encumbrance**

##### Orchestration

* Data serving is the last stage of the DE lifecycle, but b/c **serving is downstream of so many processes, it’s an area of extremely complex overlap**
* **Orchestration is not simply a way of organizing + automating complex work, but a means of coordinating data flow across teams so that data is made available to consumers at the promised time**
* **Ownership of orchestration is a key organizational decision**
* **Will orchestration be centralized or decentralized?**
* **A decentralized approach allows small teams to manage their data flows, but it can increase the burden of cross-team coordination**
* Instead of simply managing flows w/in a single system, directly triggering the completion of DAGs or tasks belonging to other teams, teams must pass messages or queries between systems.
* **A centralized approach means that work is easier to coordinate, but significant gatekeeping must also exist to protect a single production asset**
* Ex: A poorly written DAG can bring Airflow to a halt
* The centralized approach would mean bringing down data processes + serving across the whole organization
* **Centralized orchestration management requires high standards, automated testing of DAGs, + gatekeeping**
* If orchestration *IS* centralized*, who will own it?*
* **When a company has a DataOps team, orchestration usually lands here**
* **Often, a team involved in serving is a natural fit b/c it has a fairly holistic view of all DE lifecycle stages**
* This could be the DBAs, analytics engineers, DE’s, or MLE’s
* MLE’s coordinate complex model-training processes but may or may not want to add the operational complexity of managing orchestration to an already crowded docket of responsibilities

##### Software Engineering

* **Compared to a few years ago, serving data has become simpler**
* **The need to write code has been drastically simplified**
* **Data has also become more code-first, w/ the proliferation of OSS frameworks focused on simplifying the serving of data**
* **Many ways exist to serve data to end users, + a DE’s focus should be on knowing how these systems work + how data is delivered.**
* Despite the simplicity of serving data, **if code is involved, a DE should still understand how the main serving interfaces work**
* Ex: A DE may need to translate code a data scientist is running locally on a notebook + convert it into a report or a basic ML model to operate
* **Another area where DE will be useful is understanding the impact of how code + queries will perform against the storage systems**
* Analysts can generate SQL in various programmatic ways, including LookML, Jinja via dbt, various object-relational mapping (ORM) tools, + metrics layers
* **When these programmatic layers compile to SQL, how will this SQL perform?**
* **A DE can suggest optimizations where the SQL code might not perform as well as handwritten SQL**
* **The rise of analytics + ML IaC means the role of writing code is moving toward building the systems that support data scientists and analysts**
* **DE’s might be responsible for setting up the CI/CD pipelines + building processes for their data team**
* They would **also do well to train + support their data team in using the Data/MLOps infrastructure they’ve built so that these data teams can be as self-sufficient as possible**
* **For *embedded* analytics, DE’s may need to work w/ application developers to ensure that queries are returned quickly and cost-effectively**
* The application developer will control the frontend code that users deal with, + the **DE is there to ensure that developers receive the correct payloads as they’re requested**

#### Conclusion

* **The DE lifecycle has a logical ending at the serving stage, + as w/ all lifecycles, a feedback loop occurs**
* You should **view the serving stage as a chance to learn what’s working and what can be improved**
* **Listen to your stakeholders**
* If they bring up issues (+ they inevitably will), try not to take offense
* Instead, **use this as an opportunity to improve what you’ve built**
* **A good DE is always open to new feedback + constantly finds ways to improve their craft**

#### Additional Resources

* “Data as a Product vs. Data Products: What Are the Differences?” by Xavier Gumara Rigol
* “Data Jujitsu: The Art of Turning Data into Product” by D. J. Patil
* *Data Mesh* by Zhamak Dehghani (O’Reilly)
* “Data Mesh Principles and Logical Architecture” by Zhamak Dehghani
* “Designing Data Products” by Seth O’Regan
* “The Evolution of Data Products” and “What Is Data Science” by Mike Loukides
* Forrester’s “Self-Service Business Intelligence: Dissolving the Barriers to Creative Decision-Support Solutions” blog article
* “Fundamentals of Self-Service Machine Learning” by Paramita (Guha) Ghosh
* “The Future of BI Is Headless” by ZD
* “How to Build Great Data Products” by Emily Glassberg Sands
* “How to Structure a Data Analytics Team” by Niall Napier
* “Know Your Customers’ ‘Jobs to Be Done’” by Clayton M. Christensen et al.
* “The Missing Piece of the Modern Data Stack” and “Why Is Self-Serve Still a Problem?” by Benn Stancil
* “Self-Service Analytics” in the Gartner Glossary
* Ternary Data’s “What’s Next for Analytical Databases? w/ Jordan Tigani (Mother‐Duck)” video
* “Understanding the Superset Semantic Layer” by Srini Kadamati
* “What Do Modern Self-Service BI and Data Analytics Really Mean?” by Harry Dix
* “What Is Operational Analytics (and How Is It Changing How We Work with Data)?” by Sylvain Giuliani
* “What Is User-Facing Analytics?” by Chinmon Soman