# Fundamentals of Data Engineering - Reis & Housley

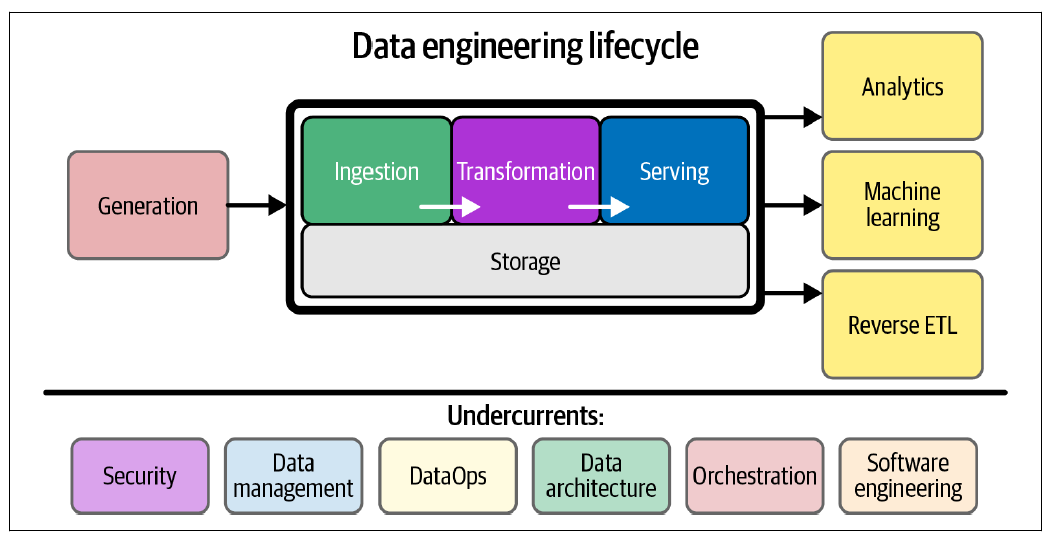
## Part I. Foundation and Building Blocks

### Chapter 2 - The Data Engineering Lifecycle

* **It’s best to move beyond viewing DE as a specific collection of data technologies**
* The data landscape is undergoing an explosion of new tech + practices, w/ ever-increasing levels of abstraction + ease of use
* ***Because of increased technical abstraction*, DE’s will increasingly become** **data lifecycle engineers**, thinking and operating in terms of the ***principles* of data lifecycle management**
* The **data engineering lifecycle** is our **framework describing “cradle to grave”** **DE**, along w/ the **undercurrents** of the lifecycle, which are **key foundations that support all DE efforts**

#### What Is the Data Engineering Lifecycle?

* **DE lifecycle** comprises **5 stages (+ 7 major undercurrents)** that **turn raw data ingredients into a useful end product, ready for consumption by analysts, data scientists, MLE’s, etc.: Generation, Storage, Ingestion, Transformation, Serving**



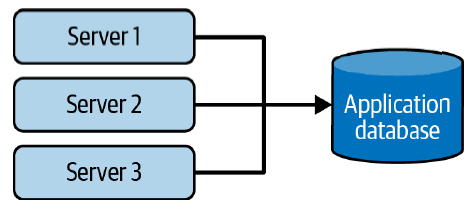
* We begin the DE lifecycle by **getting data from source systems and storing it**
* Next, we **transform** the data and then proceed to our **central goal**: **serving data** to analysts, data scientists, MLE’s, + others
* **In reality, storage occurs *throughout* the lifecycle as data flows from beginning to end**
* Hence, the above diagram shows a storage “stage” as a **foundation** that underpins other stages
* In general, the middle stages (storage, ingestion, transformation) can get a bit jumbled, + that’s OK
* **Although we split out the distinct parts of the DE lifecycle, it’s not always a neat, continuous flow**
* Various stages of the lifecycle **may repeat themselves**, occur **out of order**, **overlap**, or **weave together** in interesting and unexpected ways.
* Acting as a bedrock are **undercurrents**that cut across multiple stages of the DE lifecycle: **security, data management, DataOps, data architecture, orchestration, + SWE**
* ***No part of the data engineering lifecycle can adequately function w/out these undercurrents***

##### The Data Lifecycle Vs. The Data Engineering Lifecycle

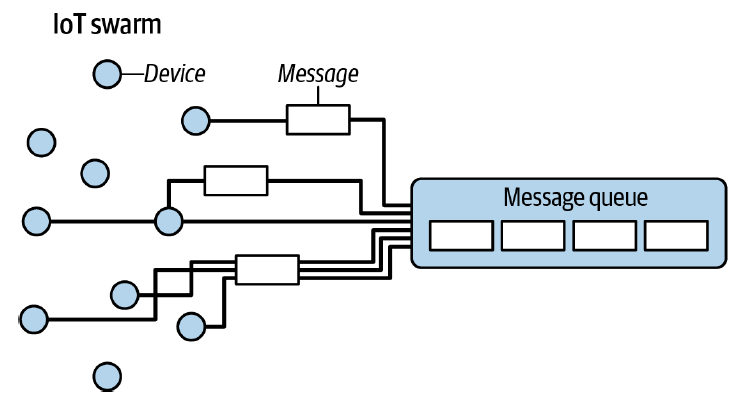
* There’s a subtle distinction between the *overall* data lifecycle and the DE lifecycle
* The **DE lifecycle is a *subset* of the whole data lifecycle**
* **Whereas the full data lifecycle encompasses data across its entire lifespan, the DE lifecycle focuses on the stages a DE can control**

##### 1) Generation: Source Systems

* A **source system**is **the origin of the data used in the DE lifecycle**
* Could be an IoT device, an application message queue, or a transactional database, etc.
* A DE *consumes* data from a source system but *doesn’t* typically *own* nor *control* the source system itself
* A **DE needs to have a working understanding of the way source systems work, the way they generate data, the frequency + velocity of the data, + the variety of data they generate**
* DE’s **also need to keep an open line of communication w/ source system owners on changes that could break pipelines + analytics**
* Application code might change the structure of data in a field, or the application team might even choose to migrate the backend to an entirely new database technology
* A **major challenge in DE is the dizzying array of source systems DE’s must work with + understand**
* 2 common source systems are a very traditional one (an application database) + the other is a more recent example (IoT swarms)
* The figure below illustrates a **traditional source system w/ several application servers supported by a database**



* This source system pattern became popular in the 1980s w/ the explosive success of relational database management systems (RDBMSs)
* **The application + database pattern remains popular today w/ various modern evolutions of software development practices**
* Ex: Applications often consist of many small service/database pairs w/ microservices rather than a single monolith.
* Below illustrates an **IoT swarm: a fleet of devices (circles) sends data messages (rectangles) to a central collection system**
* This IoT source system is increasingly common as IoT devices such as sensors, smart devices, + much more increase in the wild



* There are **many things to consider when assessing source systems**, including **how the system handles ingestion, state, + data generation**
* The following is a **starting set of evaluation questions of source systems** that DE must consider:
* **What are the essential characteristics of the data source?**
* Is it an application? A swarm of IoT devices?
* **How is data persisted in the source system?**
* Is data persisted long term, or is it temporary + quickly deleted?
* **At what rate is data generated?**
* How many events per second? How many GB per hour?
* **What level of consistency can DE’s expect from the output data?**
* **If running data-quality checks against the output data, how often do data inconsistencies occur** (NULLs where they aren’t expected, lousy formatting, etc.)?
* **How often do errors occur?**
* **Will the data contain duplicates?**
* **Will some data values arrive late, possibly much later than other messages produced simultaneously?**
* **What is the schema of the ingested data?**
* Will DE’s need to JOIN across several tables or even several systems to get a complete picture of the data?
* **If schema changes** (say, a new column is added)**, how is this dealt with + communicated to downstream stakeholders?**
* **How frequently should data be pulled from the source system?**
* **For stateful systems (e.g., a database tracking customer account information), is data provided as periodic snapshots or update events from change data capture (CDC)?**
* What’s the **logic for how changes are performed**, + **how are these tracked in the source database?**
* **Who/what is the data provider that will transmit the data for downstream consumption?**
* **Will reading from a data source impact its performance?**
* **Does the source system have upstream data dependencies?**
* What are the **characteristics of these upstream systems?**
* **Are data-quality checks in place to check for late or missing data?**
* **Sources produce data consumed by *downstream* systems**, including human-generated spreadsheets, IoT sensors, and web + mobile apps
* **Each source has its unique volume + cadence of data generation**
* A **DE should know how the source generates data, including relevant quirks or nuances**, + **also need to understand the limits of the source systems they interact w/**
* Ex: Will analytical queries against a source application database cause resource contention + performance issues?
* **One of the most challenging nuances of source data is the schema**
* The **schema****defines the hierarchical organization of data**
* Logically, we can think of data at the level of a whole source system, drilling down into individual tables, all the way to the structure of respective fields
* The **schema of data shipped from source systems is handled in various ways**
* 2 popular options are **schema-less** and **fixed schema**
* **Schema-less *doesn’t* mean the *absence* of schema,** but **rather means that the application defines the schema *as data is written***, whether to a message queue, a flat file, a blob, or a document database such as MongoDB
* **Fixed schema** = a more traditional model built on RDB storage, **enforced in the database, to which application writes *must* conform**
* **Either of these models presents different challenges for DE’s**
* **Schemas change over time**
* **In fact, schema evolution is *encouraged* in the Agile approach to SWE**
* A **key part of the DE’s job is taking raw data input in the source system schema + transforming this into valuable output for analytics, + *this job becomes more challenging as the source schema evolves***

##### 2) Storage

* You **need a place to store data**
* **Choosing a storage solution is key to success in the rest of the data lifecycle, + it’s also one of the most complicated stages of the data lifecycle for a variety of reasons**
* **1) Data architectures in the cloud often leverage *several* storage solutions**
* **2) Few data storage solutions function *purely* as storage**
* With many supporting complex transformation queries (Even object storage solutions may support powerful query capabilities, like Amazon S3 Select)
* **3) While storage is *one* stage of the DE lifecycle, it frequently touches on *other* stages, such as ingestion, transformation, + serving**
* **Storage runs across the entire DE lifecycle**, often occurring in **multiple places** in a data pipeline, with **storage systems crossing over with source systems, ingestion, transformation, + serving**
* **In many ways, the way data is stored impacts how it is used in *ALL* of the stages of the DE lifecycle**.
* Ex: Cloud DW’s can store data, process data in pipelines, + serve it to analysts
* Ex: Streaming frameworks such as Apache Kafka + Pulsar can function simultaneously as ingestion, storage, + query systems for messages, w/ object storage being a standard layer for data transmission
* Here are **a few key engineering questions to ask when choosing a storage system** for a DW, data lakehouse, database, or object storage:
* **Is this storage solution compatible with the architecture’s required write + read speeds?**
* **Will storage create a bottleneck for downstream processes?**
* **Do you *understand* how this storage technology works?**
* Are you **utilizing the storage system optimally or committing unnatural acts?**
* Example: Are you applying a high rate of random access updates in an object storage system? (This is an **antipattern (instead of a solution, it gives something that looks superficially like a solution but isn't one)** w/ significant performance overhead)
* **Will this storage system handle anticipated future scale?**
* You should **consider *all* capacity limits on the storage system: total available storage, read operation rate, write volume, etc.**
* **Will downstream users + processes be able to retrieve data in the required service-level agreement (SLA)?**
* **Are you capturing metadata about schema evolution, data flows, data lineage, + so forth?**
* **Metadata has a significant impact on the utility of data** + **represents an investment in the *future*, dramatically enhancing discoverability + institutional knowledge to streamline future projects + architecture changes**
* **Is this a *pure* storage solution (Ex: object storage), or does it also support complex query patterns (Ex: a cloud DW)?**
* **Is the storage system schema-agnostic (object storage)? Flexible schema (Cassandra)? Enforced schema (a cloud DW)?**
* **How are you tracking master data, golden records data quality, + data lineage for data governance?**
* **How are you handling regulatory compliance and data sovereignty?**
* Ex: Can you store your data in certain geographical locations but not others?

###### Understanding data access frequency

* **Not all data is accessed in the same way: Retrieval patterns will greatly vary based on the data being stored + queried**
* This brings up the notion of the “**temperatures**” of data
* **Data access frequency will determine the temperature of your data**
* **Data that is most frequently accessed is hot data**
* Commonly **retrieved many times per day, perhaps several times per second** (ex: in systems that serve user requests)
* **This data should be stored for fast retrieval, where “fast” is relative to the use case**
* **Lukewarm datamight be accessed every so often (say, every week or month)**
* **Cold data**is **seldom queried + is appropriate for storing in an archival system**
* Often retained for compliance purposes or in case of a catastrophic failure in another system
* In the “old days,” cold data would be stored on tapes + shipped to remote archival facilities, but now **w/ cloud environments, vendors offer specialized storage tiers w/ very cheap monthly storage costs but high prices for data retrieval**

###### Selecting a storage system

* **What type of storage solution you should use depends on your use cases, data volumes, frequency of ingestion, format, + size of the data being ingested**
* **There is NO one-size-fits-all universal storage recommendation**
* ***Every* storage technology has its trade-offs**
* Countless varieties of storage tech exist, + it’s **easy to be overwhelmed** when deciding the best option for *your* data architecture

##### 3) Ingestion

* After you understand the data source, the characteristics of the source system you’re using, + how data is stored, you **need to gather the data**
* The next stage of the DE lifecycle is **data ingestion *from source systems***
* **Source systems + ingestion represent the most significant bottlenecks of the DE lifecycle**
* The **source systems are normally outside your direct control + might randomly become unresponsive or provide data of poor quality**
* Or your **data ingestion service might mysteriously stop working for many reasons**
* As a result, data flow stops or delivers insufficient data for storage, processing, + serving.
* **Unreliable source + ingestion systems have a ripple effect *across* the DE lifecycle**
* But, **assuming you’ve answered the big questions about source systems, you’re in good shape**
* **When preparing to architect or build a system, here are some primary questions about the ingestion stage:**
* **What are the use cases for the data I’m ingesting?**
* Can I ***reuse* this data rather than create multiple versions** of the same dataset?
* **Are systems generating + ingesting this data *reliably*, + is the data available when I need it?**
* **What is the data destination after ingestion?**
* **How frequently will I need to access the data?**
* **In what volume will the data typically arrive?**
* **What format is the data in?**
* **Can my downstream storage + transformation systems handle this format?**
* **Is the source data in good shape for *immediate* downstream use?**
* **If so, for how long, + what may cause it to be unusable?**
* **If the data is from a streaming source, does it need to be *transformed* before reaching its destination?**
* **Would an in-flight transformation be appropriate, where the data is transformed w/in the stream itself?**
* These are just a sample of the factors you’ll need to think about w/ ingestion
* Now, let’s briefly turn our attention to **2 major data ingestion concepts: batch vs. streaming and push vs. pull**

###### Batch versus streaming

* **Virtually *all* data we deal w/ is *inherently* streaming**
* **Data is nearly always produced + updated continually at its source**
* **Batch ingestionis simply a specialized and convenient way of processing this stream in large chunks** (Ex: Handling a full day’s worth of data in a single batch)
* **Streaming ingestion allows us to provide data to downstream systems (other applications, databases, analytics systems, etc.) in a continuous, real-time fashion**
* Here, **real-time(or near real-time) means the data is available to a downstream system a short time after it’s produced (e.g., < 1 second later)**
* The **latency required to qualify as “real-time” varies by domain + requirements**
* **Batch data is ingested either on a predetermined time interval or as data reaches a preset size threshold**
* **Batch ingestion is a one-way door: once data is broken into batches, the latency for downstream consumers is inherently constrained**
* **Because of limitations of legacy systems, batch was for a long time the default way to ingest data**
* **Batch processing remains an extremely popular way to ingest data for downstream consumption, particularly in analytics + ML**
* However, **the separation of storage + compute in many systems + the ubiquity of event-streaming + processing platforms make the continuous processing of data streams much more accessible + increasingly popular**
* The ***choice largely depends on the use case + expectations for data timeliness***
* *Should you go streaming-first?*
* **Despite the attractiveness of a streaming-first approach, there are *many* trade-offs to understand + think about**
* The following are **some questions to ask yourself when determining whether streaming ingestion is an appropriate choice over batch ingestion:**
* **If I ingest the data in real time, can downstream storage systems handle the rate of data flow?**
* **Do I even *need* millisecond real-time data ingestion?**
* Or would a **micro-batch approach** work, accumulating + ingesting data, say, every minute?
* **What are my use cases for streaming ingestion?**
* What ***specific* benefits do I realize by implementing streaming?**
* If I get data in real time, **what actions can I take on that data that would be an improvement upon batch?**
* **Will my streaming-first approach cost more in terms of time, money, maintenance, downtime, + opportunity cost than simply doing batch?**
* **Are my streaming pipeline and system reliable + redundant if infrastructure fails?**
* **What tools are most appropriate for the use case?**
* **Should I use a managed service** (Amazon Kinesis, Google Cloud Pub/Sub, Google Cloud Dataflow) **or stand up my own instances** of Kafka, Flink, Spark, Pulsar, etc.?
* **If I do the latter, who will manage it?**
* **What are the costs + trade-offs?**
* **If deploying an ML model, what benefits do I have with online predictions + possibly continuous training?**
* **Am I getting data from a live production instance?**
* If so, **what’s the impact of my ingestion process on this source system?**
* **So, streaming-first might seem like a good idea, but it’s not always straightforward; extra costs + complexities inherently occur**
* **Many great ingestion frameworks do handle *both* batch + micro-batch ingestion styles**
* **Batch is an excellent approach for many common use cases, such as model training + weekly reporting**
* ***Adopt true real-time streaming only after identifying a business use case that justifies the trade-offs against using batch***

###### Push versus pull

* In **the pushmodel of data** **ingestion, a source system writes data out to a target, whether a database, object store, or filesystem**
* In **the pullmodel, data is retrieved from the source system**
* **The line between the push and pull paradigms can be quite blurry; data is often pushed *and* pulled as it works its way through the various stages of a data pipeline**
* Consider, for example, an ETL process, commonly used in batch-oriented ingestion workflows
* **ETL’s extract(E) part clarifies that we’re dealing with a *pull* ingestion model**
* In traditional ETL, the ingestion system queries a current source table snapshot on a fixed schedule
* In another example, consider **continuous CDC**, which is **achieved in a few ways**
* One common method **triggers a message every time a row is changed in the source database.**
* This **message is *pushed* to a queue**, where the **ingestion system picks it up**
* Another common CDC method uses **binary logs, which record every commit to the database**
* The **database *pushes* to its logs** + the **ingestion system reads the logs but *doesn’t directly interact w/ the database otherwise***, which adds little to no additional load to the source database
* **Some versions of batch CDC use the pullpattern**
* Ex: In timestamp-based CDC, an ingestion system queries the source database + pulls the rows that have changed since the previous update
* **W/ streaming ingestion, data bypasses a backend database + is pushed directly to an endpoint, typically w/ data buffered by an event-streaming platform**
* This pattern is useful w/ fleets of IoT sensors emitting sensor data
* **Rather than relying on a database to maintain the current state, we simply think of each recorded reading as an event**
* This pattern is also growing in popularity in software applications as it **simplifies real-time processing, allows app developers to tailor their messages for downstream analytics, + greatly simplifies the lives of DE’s**

##### 4) Transformation

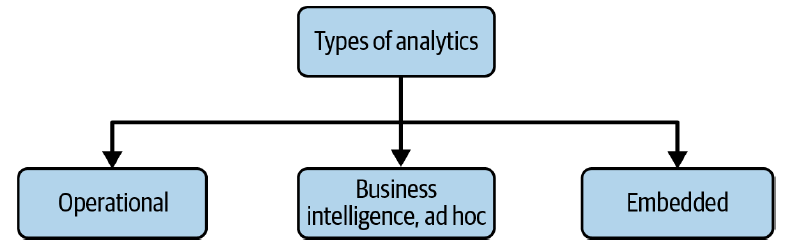
* After you’ve ingested and stored data, you **need to do something with it**
* The next stage of the DE lifecycle is **transformation** **= data needs to be changed from its original form into something *useful* for downstream use cases**
* **W/out proper transformations, data will sit inert, and not be in a useful form for reports, analysis, or ML**
* **Typically, the transformation stage is where data begins to create value for downstream user consumption**
* I**mmediately after ingestion, basic transformations map data into correct types** (ex: changing ingested string data into numeric + date types), putting records into standard formats, + removing bad ones
* **Later stages of transformation may transform the data schema** + apply **normalization**
* Downstream, we can apply **large-scale aggregation for reporting or featurize data for ML processes**
* **When considering data transformations within the DE lifecycle, it helps to consider the following:**
* **What’s the cost + ROI of the transformation? What is the associated business value?**
* **Is the transformation as simple + self-isolated as possible?**
* **What business rules do the transformations support?**
* You can **transform data in batch or while streaming in flight**
* Remember, **virtually ALL data starts life as a continuous stream + batch is just a specialized way of processing a data stream**
* **Batch transformations are overwhelmingly popular, but given the growing popularity of stream-processing solutions + the general increase in the amount of streaming data, expect the popularity of streaming transformations to continue growing, perhaps entirely replacing batch processing in certain domains soon**
* ***Logically*, we treat transformation as a standalone area of the DE lifecycle, but the realities of the lifecycle can be much more complicated in practice.**
* **Transformation is often entangled in other phases of the lifecycle**
* **Typically, data is transformed in source systems or in-flight during ingestion**
* Ex: A source system may add an event timestamp to a record before forwarding it to an ingestion process
* Ex: Or a record within a streaming pipeline may be “enriched” with additional fields + calculations before it’s sent to a DW
* **Transformations are ubiquitous in various parts of the lifecycle**
* **Data prep, wrangling, + cleaning are transformative tasks that add value for end consumers of data**
* **Business logic is a major driver of data transformation, often in data modeling**
* **Data translates business logic into reusable elements**
* Ex: a “sale” means “somebody bought 12 picture frames from me for $30 each, or $360 in total”
* **Data modeling is critical for obtaining a clear + current picture of business processes**
* A **simple view of raw retail transactions might not be useful w/out adding the logic of accounting rules** so that the CFO has a clear picture of financial health
* **Ensure a standard approach for implementing business logic across your transformations**
* **Data featurization for ML is another data transformation process**
* **Featurization intends to extract + enhance data features useful for training ML models**
* Featurization can be a dark art, **combining domain expertise** (to identify which features might be important for prediction) **with extensive experience in data science**
* **The main point is that once data scientists determine how to featurize data, featurization processes can be automated by DE’s in the transformation stage of a data pipeline**

##### 5) Serving Data

* Last stage of the DE lifecycle = now that data has been ingested, stored, + transformed into coherent + useful structures, **it’s time to “get” value from your data**
* “Getting” **means different things to different users**
* **Data has valuewhen it’s used for *practical* purposes**
* **Data that is not consumed or queried is simply inert**
* **Data vanity projects are a major risk for companies**
* Many companies pursued vanity projects in the big data era, gathering massive datasets in data lakes that were never consumed in any useful way
* The cloud era is triggering a new wave of vanity projects built on the latest DW’s, object storage systems, + streaming technologies
* **Data projects *must* be *intentional* across the lifecycle**
* **What is the ultimate business purpose of the data so carefully collected, cleaned, + stored?**
* Data serving is perhaps the most exciting part of the DE lifecycle + is where the magic happens and is where MLE’s can apply the most advanced techniques
* **Some of the popular uses of data: analytics, ML, + reverse ETL**

###### Analytics

* **Analytics is the core of most data endeavors**
* Once the data is stored + transformed, you’re **ready to generate reports or dashboards + do ad hoc analysis on it**
* Whereas the bulk of analytics used to encompass **BI**, it now includes other facets such as **operational** **analytics** + **embedded analytics**



* **Business Intelligence (BI)**
* BI marshals collected data to **describe a business’s past + current state**
* BI **requires using business logic to process raw data**
* ***Note that data serving for analytics is yet another area where the stages of the DE lifecycle can get tangled***
* As mentioned earlier, **business logic is often applied to data in the *transformation* stage of the DE lifecycle, but a logic-on-read approach has become increasingly popular**
* **Data is *stored* in a clean but fairly raw form, w/ minimal post-processing business logic**
* A **BI system maintains a repository of business logic + definitions**
* This **business logic is used to query the DW so that reports + dashboards align w/ business definitions**
* **As a company grows its data maturity, it will move from ad hoc data analysis to self-service analytics, allowing democratized data access to business users w/out needing IT to intervene**
* **The capability to do self-service analytics assumes that the data is good enough that people across the organization can simply access it themselves, slice + dice it however they choose, + get immediate insights**
* Although **self-service analytics is simple in theory, it’s tough to pull off in practice**
* The **main reason is that poor data quality, organizational silos, + a lack of adequate data skills often get in the way of allowing widespread use of analytics**
* **Operational Analytics**
* ***Operational* analytics focuses on the fine-grained details of *operations*, promoting actions that a user of the reports can act upon immediately**
* Could be a live view of inventory or real-time dashboarding of website or application health
* In this case, **data is consumed in real time**, either directly from a source system or from a streaming data pipeline
* The types of insights in operational analytics differ from traditional BI since **operational analytics is focused on the present + doesn’t necessarily concern historical trends**
* **Embedded Analytics**
* **Embedded (customer-facing) analytics** is **separate from BI** because, in practice, analytics provided to customers on a SaaS platform come w/ a **separate set of requirements + complications**
* *Internal* BI faces a limited audience + generally presents a limited number of unified views
* Access controls are critical but not particularly complicated
* Access is managed using a handful of roles and access tiers
* **W/ embedded analytics, the request rate for reports, + the corresponding burden on analytics systems, goes up dramatically**
* ***Access control is significantly more complicated + critical***
* Businesses may be serving separate analytics + data to thousands or more customers
* **Each customer must see their data + *only* their data**
* An internal data-access error at a company would likely lead to a procedural review
* A data leak between customers would be considered a massive breach of trust, leading to media attention + a significant loss of customers
* **Minimize your blast radius related to data leaks + security vulnerabilities**
* **Apply tenant- or data-level security w/in your storage + anywhere there’s a possibility of data leakage**

Multitenancy

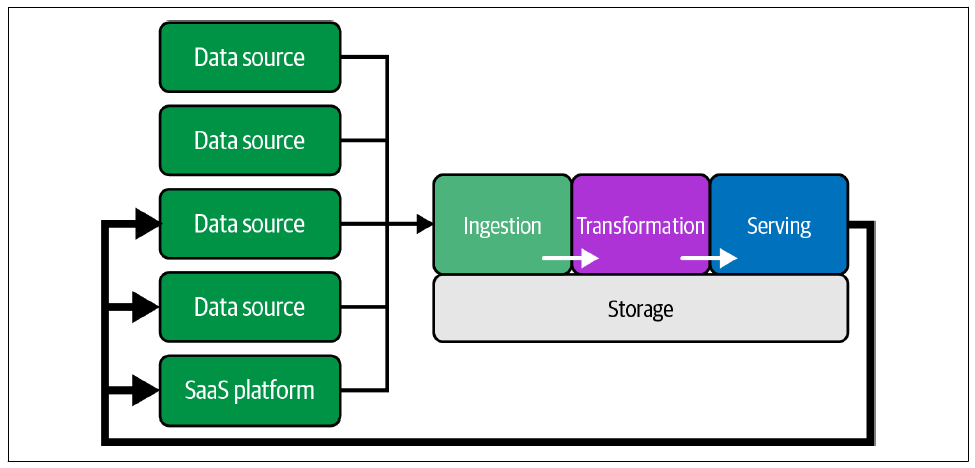
* Many current storage + analytics systems support **multitenancy (a mode of operation of software where multiple independent instances of one or multiple applications operate in a shared environment/a software architecture in which a single application serves multiple customers/tenants)** in various ways
* DE’s may choose to **house data for many customers in common tables to allow a unified view for internal analytics + ML**
* This **data is presented externally to individual customers through logical views w/ appropriately defined controls + filters**
* **It is incumbent on DE’s to understand the minutiae of multitenancy in the systems they deploy to ensure absolute data security + isolation**

###### Machine Learning

* The emergence + success of ML is one of the most exciting tech revolutions.
* *Once organizations reach a high level of data maturity*, they can begin to identify problems amenable to ML + start organizing a practice around it.
* **Responsibilities of DE’s overlap significantly in analytics + ML, + the boundaries between DE, MLE, + analytics engineering can be fuzzy**
* Ex: a DE may need to support Spark clusters that facilitate analytics pipelines + ML model training
* They may *also* need to provide a system that orchestrates tasks across teams and support metadata + cataloging systems that track data history + lineage
* **Setting these domains of responsibility + the relevant reporting structures is a critical organizational decision**
* The **feature store** is a **recently developed tool that combines DE + MLE**
* **Feature stores** are **designed to reduce the operational burden for MLE’s by maintaining feature history + versions, supporting feature sharing among teams, + providing basic operational + orchestration capabilities, such as backfilling**
* ***In practice*,** **DE’s are part of the core support team for feature stores to support MLE**
* *Should a DE be familiar with ML?* **It certainly helps**
* Regardless of the operational boundary between DE, MLE, business analytics, + so forth, **DE’s should maintain operational knowledge about their teams**
* **A good DE is conversant in the fundamental ML techniques + related data-processing requirements, the use cases for models w/in their company, + the responsibilities of the organization’s various analytics teams**
* This **helps maintain efficient communication + facilitate collaboration**
* **Ideally, DE will build tools in partnership w/ other teams that neither team can make independently**
* The following are **some considerations for the serving data phase specific to ML**:
* **Is the data of sufficient quality to perform reliable feature engineering?**
* **Quality requirements + assessments are developed in close collaboration w/ teams consuming the data**
* **Is the data discoverable? Can data scientists + MLE’s easily find valuable data?**
* **Where are the technical + organizational boundaries between DE + MLE?**
* This organizational question **has significant architectural implications**
* **Does the dataset properly represent ground truth? Is it unfairly biased?**
* **While ML is exciting, companies often prematurely dive into it**
* **Before investing a ton of resources into ML, take the time to build a solid data foundation** by **setting up the best systems + architecture across the DE + ML lifecycle**
* **Generally best to develop competence in analytics before moving to ML**
* Many companies have dashed their ML dreams because they undertook initiatives w/out appropriate foundations

###### Reverse ETL

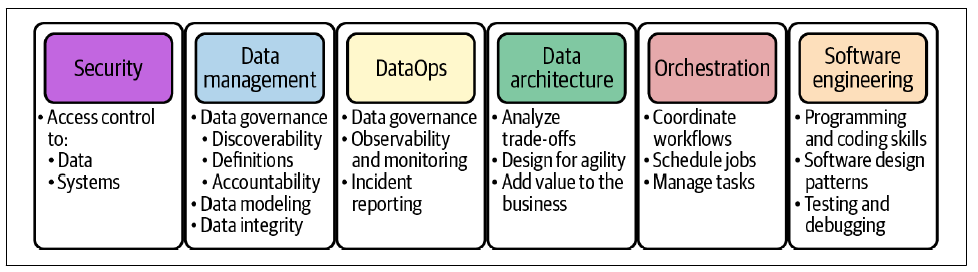
* **Reverse ETL has long been a practical reality in data, viewed as an antipattern**
* **Reverse ETL****takes processed data from the output side of the DE lifecycle + feeds it back into source systems**



* In reality, **this flow is beneficial + often necessary**
* **Reverse ETL allows us to take analytics, scored models, etc., + feed these back into production systems or SaaS platforms**
* Marketing analysts might calculate bids in Excel by using the data in their DW, + then upload these bids to Google Ads
* This process was often entirely manual + primitive
* So far (2022-2023), **several vendors have embraced the concept of reverse ETL + built products around it**, such as Hightouch and Census
* Reverse ETL remains **nascent as a practice, but it is suspected that it is here to stay.**
* Reverse ETL has **become especially important as businesses rely increasingly on SaaS + external platforms**
* Ex: Companies may want to push specific metrics from their DW to a customer data platform or CRM system
* Ex: Advertising platforms are another everyday use case, as in the Google Ads example
* Expect to see more activity in reverse ETL, with an overlap in both DE + MLE
* Though the jury is out on whether the term “reverse ETL*”* will even stick, + the practice may evolve
* Some engineers claim that we can eliminate reverse ETL by handling data transformations in an event stream + sending those events back to source systems as needed
* Realizing widespread adoption of this pattern across businesses is another matter
* The **gist is that transformed data will need to be returned to source systems in some manner, ideally w/ the correct lineage + business process associated w/ the source system**

#### Major Undercurrents Across the Data Engineering Lifecycle

* **DE is rapidly maturing**
* Whereas **prior cycles of DE simply focused on the tech layer**, the **continued abstraction + simplification of tools + practices have shifted this focus**
* **DE now encompasses *far more than tools and tech***, and **the field is now moving up the value chain, incorporating traditional enterprise practices such as data management, cost optimization, newer practices like DataOps**
* These practices (**security, data management, DataOps, data architecture, orchestration, SWE**) can be called **undercurrents**, which **support every aspect of the DE lifecycle**



##### 1) Security

* **Security *must* be top of mind for DE’s, + those who ignore it do so at their peril**
* **DE’s must understand *both* data *and* access security, whilst exercising the principle of least privilege: giving a user/system access to *only* the essential data + resources to perform an intended function**
* A **common antipattern seen w/ DE’s w/ little security experience is to give admin access to all users,** a catastrophe waiting to happen
* ***Give users only the access they need to do their jobs today, nothing more***
* Don’t operate from a root shell when just looking for visible files w/ standard user access
* When querying tables w/ a lesser role, don’t use the superuser role in a database
* **Imposing the principle of least privilege on ourselves can prevent accidental damage + keep you in a security-first mindset.**
* **People and organizational structure are always the biggest security vulnerabilities in any company**
* When hearing about major security breaches, it often turns out that someone in the company ignored basic precautions, fell victim to a phishing attack, or otherwise acted irresponsibly
* **1st line of defense for data security = create a culture of security to permeate the organization**
* All individuals who have access to data must understand their responsibility in protecting the company’s sensitive data + its customers.
* **Data security is also about *timing*: providing data access to exactly the people + systems that need to access it + *only for the duration necessary to perform their work***
* **Data should be protected from unwanted visibility, both in flight + at rest, by using encryption, tokenization, data masking, obfuscation, + simple, robust access controls**
* **DE’s must be competent security admins, as security falls in their domain**
* Should **understand security best practices for the cloud *and* on prem**
* **Knowledge of user and identity access management (IAM) roles, policies, groups, network security, password policies, + encryption are good places to start**

##### 2) Data Management

* You might think that “data management” sounds very corporate
* “Old school” data management practices make their way into DE + MLE, as what’s old is new again
* **Data management has been around for decades but didn’t get a lot of traction in DE until recently**
* **Data tools are becoming simpler, + there is less complexity for DE’s to manage**
* **As a result, the DE moves up the value chain toward the next rung of best practices**
* **Data best practices once reserved for huge companies (data governance, master data management (MDM), data quality management, metadata management, etc.) are now filtering down to companies of *all* sizes and maturity levels**
* DE is becoming “enterprise-y” (ultimately a great thing)
* Data Management Association International (DAMA) Data Management Body of Knowledge*:* **“*Data management is the development, execution, + supervision of plans, policies, programs, + practices that deliver, control, protect, + enhance the value of data + information assets throughout their lifecycle***
* **How it ties to DE: DE’s manage the data lifecycle, + *data management encompasses the set of best practices that DE’s will use to accomplish this task, both technically + strategically***
* W/out a framework for *managing* data, DE’s are simply technicians operating in a vacuum
* **DE’s need a broader perspective of data’s utility across the organization, from the source systems to the C-suite, and everywhere in between**
* *Why is data management important?*
* **Data management demonstrates that data is vital to daily operations, just as businesses view financial resources, finished goods, or real estate as assets**
* Data management practices **form a cohesive framework that everyone can adopt to ensure that the organization gets value from data + handles it appropriately**
* Data management has **quite a few facets, including the following**:
* **Data governance, including discoverability and accountability**
* **Data modeling and design**
* **Data lineage**
* **Storage and operations**
* **Data integration and interoperability**
* **Data lifecycle management**
* **Data systems for advanced analytics and ML**
* **Ethics and privacy**

###### A) Data governance

* Data Governance: The Definitive Guide:**“Data governance is, first + foremost, a data management function to ensure the quality, integrity, security, + usability of the data collected by an organization.”**
* We can expand on that: **Data governance engages people, processes, + technologies to maximize data value across an organization while protecting data w/ appropriate security controls**
* **Effective data governance is developed w/ *intention* + is supported by the organization**
* When data governance is accidental + haphazard, the side effects can range from untrusted data to security breaches + everything in between
* **Being *intentional* about data governance will maximize the organization’s data capabilities + the value generated from data**
* Will also (hopefully) keep a company out of the headlines for questionable to downright reckless data practices
* Typical example of data governance being done poorly:
* A business analyst gets a request for a report but doesn’t know what data to use to answer the question
* May spend hours digging through dozens of tables in a transactional database, wildly guessing at which fields might be useful
* The analyst compiles a “directionally correct” report but isn’t entirely sure the report’s underlying data is accurate or sound
* The recipient of the report also questions the validity of the data
* The integrity of the analyst (*and of all data in the company’s systems*) is called into question
* The company is confused about its performance, making business planning impossible
* **Data governance = a foundation for data-driven business practices + a mission-critical part of the DE lifecycle**
* **When data governance is practiced well, people, processes, + technologies align to treat data as a key business driver** **(if data issues occur, they are promptly handled)**
* **Core categories of data governance** = **discoverability**, **security**, and **accountability**
* *Within* these core categories are **subcategories**, such as **data quality**, **metadata**, + **privacy**

i) Discoverability

* **In a data-driven company, data must be available + discoverable**
* **End users should have quick + reliable access to the data they need to do their jobs**
* **They should know where the data comes from, how it relates to other data, + what the data means**
* **Key areas of data discoverability** include **metadata management** and **master data management**

a) Metadata

* **Metadata** **is “data about data,”** + it **underpins every section of the DE lifecycle** as it is **exactly the data needed to make data discoverable + governable**
* **2 major metadata categories: autogenerated** and **human generated.**
* **Modern data engineering revolves around automation, but metadata collection is often manual + error prone**
* **Tech can assist w/ this process**, removing much of the error-prone work of manual metadata collection
* We are seeing a proliferation of **data catalogs, data-lineage tracking systems, + metadata management tools**
* Tools can crawl databases to look for relationships + monitor data pipelines to track where data comes from + where it goes
* A low-fidelity manual approach uses an internally led effort where various stakeholders crowdsource metadata collection w/in the organization
* These data management tools undercut much of the DE lifecycle
* **Metadata becomes a byproduct of data + data processes**
* However, **key challenges remain** and, **in particular, interoperability + standards are still lacking**
* ***Metadata tools are only as good as their connectors to data systems + their ability to share metadata***
* In addition, **automated metadata tools should *NOT* entirely take humans out of the loop**
* **Data has a social element; each organization accumulates social capital + knowledge around processes, datasets, + pipelines**
* Human-oriented metadata systems focus on the social aspect of metadata (something Airbnb has emphasized in various blog posts on data tools, particularly its original Dataportal concept: <https://medium.com/airbnb-engineering/democratizing-data-at-airbnb-852d76c51770>)
* **Such tools should provide a place to disclose data owners, data consumers, + domain experts**
* **Documentation + internal wiki tools provide a key foundation for metadata management, but these tools should also integrate w/ automated data cataloging**
* Ex: Data-scanning tools can generate wiki pages with links to relevant data objects
* **Once metadata systems + processes exist, DE’s can consume metadata in useful ways**
* **Metadata becomes a foundation for designing pipelines + managing data throughout the lifecycle**
* Data Management Body of Knowledge ID’ **4 main categories of metadata that are useful to DE’s:**
* **1) Business metadata relates to the way data is used in the business, including business + data definitions, data rules + logic, how + where data is used, + the data owner(s)**
* **DE uses business metadata to answer *non*-technical questions like who, what, where, how**
* Ex: A DE may be tasked with creating a data pipeline for customer sales analysis
* *But what is a customer?* Is it someone who’s purchased in the last 90 days? Or someone who’s purchased at any time the business has been open?
* A DE would use the correct data to refer to business metadata (data dictionary or data catalog) to look up how a “customer” is defined
* **Business metadata provides a DE w/ the right context + definitions to properly use data**
* **2) Technical metadata describes the data created + used by systems across the DE lifecycle**
* **Includes the data model + schema, data lineage, field mappings, + pipeline workflows**
* A **DE uses technical metadata to create, connect, + monitor various systems across the DE lifecycle**
* Some common types of technical metadata that a DE will use:
* **Pipeline metadata** (often produced/captured in orchestration systems)
* **Orchestration is a central hub that coordinates workflow across various systems**
* **Provides details of the workflow schedule, system + data dependencies, configurations, connection details, + much more**
* **Data-lineage metadata****tracks the origin + changes to data, + its dependencies, over time**
* As data flows through the DE lifecycle, it evolves through transformations + combinations w/ other data
* **Data lineage provides an audit trail of data’s evolution as it moves through various systems + workflows**
* **Schema metadata****describes the structure of data stored in a system such as a database, a DW, a data lake, or a filesystem**
* It is **one of the key differentiators across different storage systems**
* Object stores, for example, don’t manage schema metadata
* Instead, this must be managed in a **metastore**
* On the other hand, cloud DW’s manage schema metadata internally
* *These are just a few examples of technical metadata that a DE should know about, not a complete list*
* **3) Operational metadata****describes the operational results of various systems + includes statistics about processes, job IDs, application runtime logs, data used in a process, + error logs**
* A **DE uses operational metadata to determine whether a process succeeded or failed + the data involved in the process**
* Orchestration systems can provide a limited picture of operational metadata, but the latter **still tends to be scattered across many systems**
* **A need for better-quality operational metadata, + better metadata management, is a major motivation for next-generation orchestration + metadata management systems**
* **4) Reference (lookup) metadatais data used to classify other data**
* Standard examples of reference data are **internal codes, geographic codes, units of measurement, + internal calendar standards**
* Note that **much of reference data is fully managed internally, but items such as geographic codes might come from** **standard external references**
* **Reference data is essentially a standard for interpreting other data, so if it changes, this change happens slowly over time**

b) Master Data Management

* **Master data**= **data about business entities** such as employees, customers, products, + locations
* As **organizations grow larger + more complex** through organic growth + acquisitions, + collaborate w/ other businesses, **maintaining a consistent picture of entities + identities becomes more + more challenging**
* **Master data management (MDM)** = **the practice of building consistent entity definitions known as golden records**, which **harmonize entity data across an organization + w/ its partners**
* **MDM is a *business* operations process facilitated by building + deploying *technology* tools**
* Ex: An MDM team might determine a standard format for addresses, + then work w/ DE’s to build an API to return consistent addresses + a system that uses address data to match customer records across company divisions
* **MDM reaches across the *full* data cycle into operational databases**
* It ***may* fall directly under the purview of DE but is often the assigned responsibility of a dedicated team that works across the organization**
* **Even if they don’t *own* MDM, DE’s must always be aware of it, as they might collaborate on MDM initiatives**

ii) Accountability

* **Data accountability**means **assigning an individual to govern a portion of data** **who then coordinates the governance activities of other stakeholders**
* Managing data quality is tough if no one is accountable for the data in question
* Note that **people accountable for data need not be DE’s** (could be a SWE, or PM, or another role)
* In addition, the **responsible person generally doesn’t have all the resources necessary to maintain data quality**
* Instead, they **coordinate w/ all people who touch the data, including DE’s**
* Data accountability **can happen at various levels** (at the level of a table or a log stream but could be as fine-grained as a single field entity that occurs across many tables)
* Ex: An individual may be accountable for managing a customer ID across many systems
* Ex: For enterprise data management, a **data domain is the set of all possible values that can occur for a given field type**, such as in this ID example
* **May seem excessively bureaucratic + meticulous, but it can significantly affect data quality**

iii) Data quality

* Everyone in the business wants to know if they can ***trust*** the data
* **Data quality**= **the optimization of data toward the desired state**, +orbits the question, “What do you get compared w/ what you expect?”
* **Data should conform to the expectations in the business metadata** **(*Does the data match the definition agreed upon by the business?*)**
* A **DE ensures data quality across the *entire* DE lifecycle**
* Involves **performing data-quality tests, + ensuring data conformance to schema expectations, data completeness, + precision**
* Data Governance: The Definitive Guide: **data quality is defined by 3 main characteristics:**
* **1) Accuracy**
* Is the collected data factually correct? Are there duplicate values? Are the numeric values accurate?
* **2) Completeness**
* Are the records complete? Do all required fields contain valid values?
* **3) Timeliness**
* Are records available in a timely fashion?
* *Each of these characteristics is quite nuanced*
* Ex: How do we think about bots + web scrapers when dealing w/ web event data?
* If we intend to analyze the customer journey, we must have a process that lets us separate humans from machine-generated traffic
* Any bot-generated events misclassified as humanpresent data accuracy issues + vice versa
* A variety *of interesting problems arise concerning completeness and timeliness*
* Ex: In the Google paper introducing the Dataflow model, the authors give the example of an offline video platform that displays ads
* The platform downloads video + ads while a connection is present, allows a user to watch these offline, + then uploads ad view data once a connection is present again
* This data may arrive late, well after the ads are watched
* How does the platform handle billing for the ads?
* <https://static.googleusercontent.com/media/research.google.com/en//pubs/archive/43864.pdf>
* Fundamentally, this problem can’t be solved by purely technical means
* Rather, engineers will need to determine their standards for late-arriving data + enforce these uniformly, possibly with the help of various technology tools
* **Data quality sits across the boundary of human + tech problems**
* **DE’s need robust processes to collect actionable human feedback on data quality + use tech tools to detect quality issues preemptively *before downstream users ever see them***

###### B) Data Modeling and Design

* **To derive business insights from data, through business analytics + data science, the data must be in a usable form**
* **The process for converting data into a usable form is known as data modeling and design**.
* Whereas we traditionally think of data modeling as a problem for DBAs + ETL developers, **data modeling can happen almost anywhere in an organization**
* Ex: Firmware engineers develop the data format of a record for an IoT device, or web app developers design the JSON response to an API call or a MySQL table schema
* **Data modeling = became more challenging b/c of the variety of new data sources + use cases**
* For instance, *strict normalization doesn’t work well with event data*
* **Fortunately, a new generation of data tools increases the flexibility of data models, while retaining logical separations of measures, dimensions, attributes, + hierarchies**
* **Cloud DW’s support the ingestion of *enormous* quantities of denormalized + semi-structured data**, while **still supporting common data modeling patterns** (ex: Kimball, Inmon, + Data Vault)
* Data processing frameworks such as Spark can ingest a whole spectrum of data, from flat structured, relational records to raw unstructured text
* With the wide variety of data that DE’s must cope w/, there is a temptation to throw up our hands + give up on data modeling
* This is a terrible idea w/ harrowing consequences, made evident when people murmur of the **write once, read never (WORN) access pattern** or refer to a **data swamp**
* **DE’s need to understand modeling best practices as well as develop the flexibility to apply the appropriate level + type of modeling to the data source + use case**

###### C) Data Lineage

* As data moves through its lifecycle, **how do you know what system affected the data or what the data is composed of as it gets passed around and transformed?**
* **Data lineage** describes **the recording of an audit trail of data through its lifecycle, tracking both the systems that process the data + the upstream data it depends on**
* **Helps w/ error tracking, accountability, + debugging of data + the systems that process it**
* **Has the obvious benefit of giving an audit trail for the data lifecycle + helps with compliance**
* Ex: If a user would like their data deleted from your systems, having lineage for that data lets you know where that data is stored + its dependencies
* Data lineage has been around for a long time in larger companies w/ strict compliance standards
* However, it’s **now being more widely adopted in smaller companies as data management becomes mainstream**
* Andy Petrella’s concept of **Data Observability Driven Development (DODD)** is closely related to data lineage: <https://www.kensu.io/blog/a-guide-to-understanding-data-observability-driven-development>
* **DODD observes data all along its lineage**, + this process is **applied during development, testing, + finally production to deliver quality + conformity to expectations**

###### D) Data Integration and Interoperability

* **Data integration and interoperability**= **the process of integrating data across tools + processes**
* As we **move away from a single-stack approach to analytics** + **toward a heterogeneous cloud environment in which various tools process data on demand, integration + interoperability occupy an ever-widening swath of the DE’s job**
* **Increasingly, integration happens through general-purpose APIs rather than custom database connections**
* Ex: A data pipeline might pull data from the Salesforce API, store it to Amazon S3, call the Snowflake API to load it into a table, call the API again to run a query, + then export the results to S3 where Spark can consume them
* *All of this activity can be managed with relatively simple Python code that talks to data systems rather than handling data directly*
* **While the complexity of interacting w/ data systems has decreased, the number of systems + the complexity of pipelines has dramatically increased**
* **Engineers starting from scratch quickly outgrow the capabilities of bespoke scripting + stumble into the need for orchestration**, one of our DE lifecycle undercurrents

###### E) Data Lifecycle Management

* The advent of **data lakes** encouraged organizations to ignore data archival + destruction
* *Why discard data when you can simply add more storage ad infinitum?*
* **2 changes have encouraged engineers to pay more attention to what happens at the end of the DE lifecycle:**
* **1) Data is increasingly stored in the cloud**
* This means we have **pay-as-you-go storage costs** instead of large up-front capital expenditures for an on-prem data lake
* When every byte shows up on a monthly AWS statement, CFOs see opportunities for savings
* Cloud environments make data archival a relatively straightforward process
* **Major cloud vendors offer archival-specific object storage classes that allow long-term data retention at an extremely low cost, assuming very infrequent access** (*but* *data retrieval isn’t so cheap*)
* These storage classes also support extra policy controls to prevent accidental or deliberate deletion of critical archives
* **2) Privacy + data retention laws such as the GDPR and the CCPA require DE’s to actively manage data destruction to respect users’ “right to be forgotten”**
* DE’s must know what consumer data they retain + must have procedures to destroy data in response to requests + compliance requirements
* **Data destruction is straightforward in a cloud DW**
* SQL semantics allow deletion of rows conforming to a WHERE clause
* Data destruction was more challenging in data lakes, where write-once, read-many was the default storage pattern
* Tools like Hive ACID + Delta Lake allow easy management of deletion transactions at scale
* **New generations of metadata management, data lineage, + cataloging tools will also streamline the end of the DE lifecycle**

###### F) Ethics and Privacy

* The last several years of data breaches, misinformation, and mishandling of data make one thing clear: **data impacts people**
* Whereas **data’s ethical + privacy implications** were once considered “nice to have”, like security, they’re **now central to the general data lifecycle**
* DE’s need to do the right thing when no one else is watching, b/c everyone will be watching someday
* *How do ethics + privacy impact the DE lifecycle?*
* DE’s need to **ensure that datasets mask personally identifiable information (PII) + other sensitive information**
* Bias can be identified + tracked in datasets as they are transformed
* Regulatory requirements + compliance penalties are only growing
* Ensure that your data assets are compliant w/ a growing number of data regulations, such as GDPR and CCPA

##### 3) DataOps

* **DataOps maps best practices of Agile methodology, DevOps, + statistical process control (SPC) to data**
* Whereas **DevOps aims to improve the release + quality of *software* products**, **DataOps** does the same thing **for *data* products**
* **Data products differ from software products *because of the way data is used***
* A **software product** provides **specific functionality** and **technical features** for end users
* **By contrast, a data product is built around sound business logic + metrics, whose users make decisions or build models that perform automated actions**
* **A DE *must* understand *both* the technical aspects of building software products *and* the business logic, quality, + metrics that will create excellent data products**
* Like DevOps, DataOps borrows much from lean manufacturing + supply chain management, **mixing people, processes, + tech to reduce time to value**
* Data Kitchen (experts in DataOps):
* “DataOps is **a collection of technical practices, workflows, cultural norms, + architectural patterns** that enable:
* **Rapid innovation + experimentation delivering new insights to customers w/ increasing velocity**
* **Extremely high data quality + very low error rates**
* **Collaboration across complex arrays of people, tech, + environments**
* **Clear measurement, monitoring, and transparency of results”**
* **Lean practices** (such as lead time reduction + minimizing defects) + the **resulting improvements to quality + productivity** are things that are good to see gaining momentum both in software + data operations
* First and foremost, **DataOps is a set of cultural habits**; a **DE team needs to adopt a cycle of communicating + collaborating w/ the business, breaking down silos, continuously learning from successes + mistakes, + rapid iteration**
* **Only when these cultural habits are set in place can the team get the best results from tech + tools**
* Depending on a company’s data maturity, a DE has some options to build DataOps into the fabric of the overall DE lifecycle
* If the company has **no preexisting data infrastructure or practices**, DataOps is very much a **greenfield opportunity** that can be baked in from day 1
* With an **existing project or infrastructure** that lacks DataOps, a DE can **begin adding DataOps into workflows**
* It’s maybe **best to first start w/ observability + monitoring** to get a window into the performance of a system, **then adding in automation + incident response**
* A DE may work alongside an existing DataOps team to improve the DE lifecycle in a data-mature company
* **In all cases, a DE must be aware of the philosophy + technical aspects of DataOps**
* **DataOps** has **3 core technical elements**: **automation**, **monitoring** **+** **observability**, **incident response**

###### A) Automation

* **Automation enables reliability + consistency in the DataOps process + allows DE’s to quickly deploy new product features + improvements to existing workflows**
* DataOps automation has a similar framework + workflow to DevOps, consisting of **change management** **(environment, code, + data version control)**, **continuous integration/continuous deployment (CI/CD)**, + **configuration as code**
* *Like DevOps*, **DataOps practices monitor + maintain the reliability of technology + systems** (data pipelines, orchestration, etc.), **with the *added* dimension of checking for data quality, data/model drift, metadata integrity, + more**
* Evolution of DataOps automation w/in a hypothetical organization.
* An organization with a low level of DataOps maturity often attempts to schedule multiple stages of data transformation processes using cron jobs, which works well for a while
* As data pipelines become more complicated, several things are likely to happen”
* a) If the cron jobs are hosted on a cloud instance, the instance may have an operational problem, causing the jobs to stop running unexpectedly
* b) As the spacing between jobs becomes tighter, a job will eventually run long, causing a subsequent job to fail or produce stale data
* c) Engineers may not be aware of job failures until they hear from analysts that reports are out-of-date
* As the organization’s data maturity grows, DE’s will typically adopt an **orchestration framework**, perhaps Airflow or Dagster
* DE are aware that Airflow presents an **operational burden**, but the **benefits of orchestration eventually outweigh the complexity**
* DE’s will gradually migrate their cron jobs to Airflow jobs
* **Now, dependencies are checked before jobs run, + more transformation jobs can be packed into a given time b/c each job can start as soon as upstream data is ready rather than at a fixed, predetermined time**
* The DE team **still has room for operational improvements**
* A data scientist eventually deploys a broken DAG, bringing down the Airflow web server + leaving the data team operationally blind
* After enough such headaches, the DE team members realize that they need to stop allowing manual DAG deployments
* In the next phase of operational maturity, they adopt **automated DAG deployment** = DAGs are tested before deployment, + monitoring processes ensure that new DAGs start running properly
* In addition, DE’s block the deployment of new Python dependencies until installation is validated
* After automation is adopted, the data team is much happier and experiences far fewer headaches
* One of the tenets of the **DataOps Manifesto** (<https://dataopsmanifesto.org/en/>) is **“Embrace change.”**
* This does *NOT* mean change for the sake of change, but rather **goal-oriented change**
* At each stage of an automation journey, opportunities exist for operational improvement:
* **Even at the high level of maturity described earlier, further room for improvement remains**
* DE’s might embrace a next-generation orchestration framework that builds in better metadata capabilities
* Or they might try to develop a framework that builds DAGs automatically based on data-lineage specifications
* The main point is that **DE’s constantly seek to implement improvements in automation that will reduce their workload + increase the value that they deliver to the business**

###### B) Observability and Monitoring

* As we tell our clients, “Data is a silent killer.”
* There’re countless examples of bad data lingering in reports for months or years
* Executives may make key decisions from this bad data, discovering the error only much later
* Outcomes are usually bad + sometimes catastrophic for the business
* Initiatives are undermined + destroyed, w/ years of work wasted
* In some of the worst cases, bad data may lead companies to financial ruin
* Another horror story occurs when the *systems* that create the data for reports randomly stop working, resulting in reports being delayed by several days
* The data team doesn’t know until they’re asked by stakeholders why reports are late or producing stale information
* Eventually, various stakeholders lose trust in the capabilities of the core data team + start their own splinter teams
* The result is many different unstable systems, inconsistent reports, + silos
* **If you’re not observing + monitoring your data + the systems that *produce* the data, you’re inevitably going to experience your own data horror story**
* **Observability**, **monitoring**, **logging**, **alerting**, + **tracing** **=** **all critical to getting ahead of any problems along the DE lifecycle**
* Also incorporate **SPC** **to understand whether events being monitored are out of line + which incidents are worth responding to**
* Petrella’s **DODD** **(Data Observability-Driven Development) method** mentioned previously provides an excellent framework for thinking about data observability
* DODD is much like **test-driven development (TDD)** in SWE
* **Purpose of DODD = to give everyone involved in the data chain visibility into the data + data applications so that everyone involved in the data value chain has the ability to identify changes to the data or data applications at every step (from ingestion to transformation to analysis) to help troubleshoot or prevent data issues.**
* DODD **focuses on making data observability a first-class consideration in the DE lifecycle**

###### C) Incident Response

* A high-functioning data team using DataOps will be able to ship new data products quickly, but **mistakes *will* inevitably happen**
* Countless problems can interrupt the DE lifecycle: A system may have downtime, a new data model may break downstream reports, an ML model may become stale + provide bad predictions, etc.
* **Incident response**is about **using the automation + observability capabilities to rapidly identify root causes of an incident + resolve it as reliably + quickly as possible**
* Incident response **isn’t *just* about tech and tools** (though these are beneficial); it’s **also about open + blameless communication, both on the DE team + across the organization**
* Werner Vogels, CTO of AWS: “Everything breaks all the time.”
* **DE’s must be prepared for a disaster + ready to respond as swiftly + efficiently as possible**
* DE’s should ***proactively* find issues before the business reports them**
* Failure happens, + when stakeholders/end users see problems, they will present them, + they will be unhappy to do so
* **The feeling is different when they go to raise those issues to a team + see that they are actively being worked on to resolve already**
* Which team’s state would you trust more as an end user?
* ***Trust takes a long time to build and can be lost in minutes***
* **Incident response is as much about *retroactively* responding to incidents as *proactively* addressing them before they happen**

###### D) DataOps Summary

* At this point, **DataOps is still a work in progress**
* Practitioners have done a good job of adapting DevOps principles to the data domain + mapping out an initial vision through the DataOps Manifesto + other resources
* **DE’s would do well to make DataOps practices a high priority in all of their work**.
* The **up-front effort will see a significant long-term payoff through faster delivery of products, better reliability + accuracy of data, + greater overall value for the business**
* The state of operations in DE is still quite immature compared with SWE
* Many DE tools, especially legacy monoliths, are *NOT* automation-first
* A recent movement has arisen to adopt automation best practices across the DE lifecycle
* Tools like Airflow have paved the way for a new generation of automation + data management tools
* The general practices described for DataOps are aspirational, + companies should try to adopt them to the fullest extent possible, given the tools + knowledge available today

##### 4) Data Architecture

* A **data architecture** **reflects the current + future state of data systems that support an organization’s long-term data needs + strategy**
* **Because an organization’s data requirements will likely change rapidly, + new tools + practices seem to arrive on a near-daily basis, DE must understand *good* data architecture**
* A **DE should first understand the needs of the business + gather requirements for new use cases**
* Next, a **DE needs to translate those requirements to design new ways to capture + serve data, balanced for cost + operational simplicity**
* This means **knowing the trade-offs with design patterns, tech, + tools in source systems, ingestion, storage, transformation, + serving data**
* This ***doesn’t* imply that a DE is a data architect, as these are typically 2 *separate* roles**
* If a DE works alongside a data architect, the **DE should be able to deliver on the data architect’s designs + provide architectural feedback**

##### 5) Orchestration

* Orchestration is not only a central DataOps process, but also a critical part of the engineering + deployment flow for data jobs
* **Orchestration** = the **process of coordinating many jobs to run as quickly + efficiently as possible on a scheduled cadence**
* Ex: People often refer to orchestration tools like Apache Airflow as **schedulers**, which *isn’t quite accurate*
* A ***pure* scheduler**, such as cron, **is aware only of *time***; an **orchestration engine builds in metadata on job dependencies, generally in the form of a DAG that can be run once or scheduled to run at a fixed interval of daily,** **weekly, every hour, every 5 minutes, etc.**
* Assume that an **orchestration system stays online w/ high availability**, which **allows such a system to sense + monitor constantly w/ out human intervention + run new jobs anytime they are deployed**
* An **orchestration system monitors jobs that it manages + kicks off new tasks as internal DAG dependencies are completed**
* **Can also monitor *external* systems + tools to watch for data to arrive + criteria to be met**
* When **certain conditions go out of bounds, the system also sets error conditions + sends alerts through email or other channels**
* Ex: You might set an expected completion time of 10 a.m. for overnight daily data pipelines
* If jobs are not done by this time, alerts go out to DE’s + consumers
* **Orchestration systems also build job history capabilities, visualization, and alerting**
* *Advanced* orchestration engines can backfill new DAGs or individual tasks as they’re added to a DAG
* They also support dependencies over a time range
* Ex: A monthly reporting job might check that an ETL job has been completed for the full month before starting
* **Orchestration has long been a key capability for data processing but was not often top of mind nor accessible to anyone except the largest companies**
* Enterprises used various tools to manage job flows, but these were expensive, out of reach of small startups, + generally not extensible
* Apache Oozie was extremely popular in the 2010s, but was designed to work w/in a Hadoop cluster + was difficult to use in a more heterogeneous environment
* Facebook developed Dataswarm for internal use in the late 2000s, which inspired popular tools such as Airflow, introduced by Airbnb in 2014
* Airflow was open-source from its inception, was widely adopted, + was written in Python, making it highly extensible to almost any use case imaginable
* Many interesting open-source orchestration projects exist (Ex: Prefect, Dagster, Luigi, Conductor, etc.), but Airflow is arguably the mindshare leader for the time being
* Airflow arrived just as data processing was becoming more abstract + accessible, + engineers were increasingly interested in coordinating complex flows across multiple processors + storage systems, especially in cloud environments
* As of 2022-2023, several nascent open-source projects aim to mimic the best elements of Airflow’s core design while improving on it in key areas
* Some of the most interesting examples = **Prefect** and **Dagster**, which aim to improve the portability + testability of DAGs to allow engineers to move from local development to production more easily
* **Argo** is an orchestration engine built around Kubernetes primitives
* **Metaflow** is an OSS project from Netflix that aims to improve data science orchestration
* **Orchestration is strictly a *batch* concept**
* The **streaming alternative to orchestrated task DAGs is the** **streaming DAG**, which remain **challenging to build + maintain**
* But next-generation streaming platforms such as **Pulsar** aim to dramatically reduce the engineering + operational burden

##### 6) Software Engineering

* SWE has *always* been a central skill for DE’s
* In the early days of contemporary DE (2000–2010), DE’s worked on low-level frameworks + wrote MapReduce jobs in C, C++, and Java
* At the peak of the big data era (the mid-2010s), engineers started using frameworks that **abstracted away these low-level details**
* This **abstraction continues today**
* Cloud DW’s support powerful transformations using SQL semantics, + tools like Spark have become more user-friendly, transitioning away from low-level coding details + toward easy-to-use **dataframes**
* Despite this abstraction, **SWE is still critical to DE**
* Here are **a few common areas of SWE that apply to the DE lifecycle**:

###### A) Core Data Processing Code

* Though it has **become more abstract + easier to manage**, the **core data processing code still needs to be written, + it appears throughout the DE lifecycle**s
* **Whether in ingestion, transformation, or data serving, DE’s need to be highly proficient + productive in frameworks + languages such as Spark, SQL, or Beam**
* It’s **also *imperative* that a DE understand proper code-testing methodologies, such as unit, regression, integration, end-to-end, + smoke**

###### B) Development of Open-source Frameworks

* Many DE’s are heavily involved in developing open-source frameworks
* They **adopt these frameworks to solve specific problems in the DE lifecycle, + then continue developing the framework code to improve the tools for *their* specificuse cases + contribute back to the community**
* In the big data era, we saw a Cambrian explosion of data-processing frameworks inside the Hadoop ecosystem
* These tools primarily focused on transforming + serving parts of the DE lifecycle
* **DE tool speciation has not ceased or slowed down, but the emphasis has shifted up the ladder of abstraction, away from *direct* data processing**
* This new generation of open-source tools assists engineers in managing, enhancing, connecting, optimizing, + monitoring data
* Ex: Airflow dominated the orchestration space from 2015 until early 2020s
* Now, a new batch of open-source competitors (Prefect, Dagster, Metaflow, etc.) has sprung up to fix perceived limitations of Airflow, providing better metadata handling, portability, + dependency management
* **Before DE’s begin engineering new internal tools, they would do well to survey the landscape of publicly available tools**
* Keep an eye on the **total cost of ownership (TCO)** and **opportunity cost associated w/ implementing a tool**
* There is a good chance that an OSS project already exists to address the problem you’re looking to solve, + they would do well to collaborate rather than reinventing the wheel

###### C) Streaming

* **Streaming data processing is inherently more complicated than batch, + the tools + paradigms are arguably less mature**
* As streaming data becomes more pervasive in every stage of the DE lifecycle, DE’s face interesting SWE problems
* Ex: Data processing tasks such as **JOINs** that we take for granted in the batch processing world often **become more complicated in real time, requiring more complex SWE**
* **Engineers must also write code to apply a variety of** **windowing methods** (**allows real-time systems to calculate valuable metrics** such as trailing statistics)
* Engineers have **many frameworks to choose from**, including **various function platforms** (OpenFaaS, AWS Lambda, Google Cloud Functions) for handling individual events **or dedicated stream processors** (Spark, Beam, Flink, Pulsar) for analyzing streams to support reporting + real-time actions

###### D) Infrastructure as Code

* **Infrastructure as code (IaC)** **applies SWE practices to configuration + management of infrastructure**
* The infrastructure management burden of the big data era has decreased as companies have migrated to *managed* big data systems (Databricks, Amazon Elastic MapReduce (EMR), etc.) + cloud DW’s
* **When DE’s have to manage their infrastructure in a cloud environment, they increasingly do this through IaC frameworks rather than manually spinning up instances + installing software**
* Several general-purpose + cloud platform-specific frameworks allow automated infrastructure deployment based on a set of specifications
* Many of these frameworks can manage cloud services as well as infrastructure
* There is also a notion of IaC w/ containers + Kubernetes, using tools like Helm.
* These practices are a **vital part of DevOps, allowing version control + repeatability of deployments**
* Naturally, **these capabilities are vital throughout the DE lifecycle, especially as we adopt DataOps practices**

###### E) Pipelines as Code

* **Pipelines as code**= **the core concept of present-day orchestration systems, which touch every stage of the DE lifecycle**
* DE’s use code (typically Python) to declare data tasks + dependencies among them
* The orchestration engine interprets these instructions to run steps using available resources.

###### F) General-Purpose Problem Solving

* In practice, regardless of which high-level tools they adopt, **DE’s will run into corner cases throughout the DE lifecycle that require them to solve problems outside the boundaries of their chosen tools + to write custom code**
* When using frameworks like Fivetran, Airbyte, or Matillion, DE’s will encounter data sources w/out existing connectors + need to write something custom
* **They should be proficient in SWE to understand APIs, pull + transform data, handle exceptions, + so forth**

#### Conclusion

* Most discussions we’ve seen in the past about DE involve tech but miss the bigger picture of **data lifecycle management**
* **As tech becomes more abstract + does more heavy lifting, a DE has the opportunity to think + act on a higher level**
* The **5-stage DE lifecycle (Generation, Storage, Ingestion, Transformation, Serving)**, supported by its **undercurrents (Security, Data management, DataOps, Data architecture, Orchestration, SWE)**, is an extremely useful mental model for organizing the work of DE
* Undercurrents = several themes that cut across the DE lifecycle
* **A DE has several top-level goals across the data lifecycle: produce optimum ROI + reduce costs (financial + opportunity), reduce risk (security, data quality), and maximize data value + utility**
* **These elements impact good architecture design, along with choosing the right technologies**

#### Additional Resources

* “A Comparison of Data Processing Frameworks” by Ludovic Santos: <https://kapernikov.com/a-comparison-of-data-processing-frameworks/>
* DAMA International website: <https://www.dama.org/cpages/body-of-knowledge>
* “The Dataflow Model: A Practical Approach to Balancing Correctness, Latency, and Cost in Massive-Scale, Unbounded, Out-of-Order Data Processing” by Tyler Akidau et al: <https://static.googleusercontent.com/media/research.google.com/en//pubs/archive/43864.pdf>
* “Democratizing Data at Airbnb” by Chris Williams et al: <https://medium.com/airbnb-engineering/democratizing-data-at-airbnb-852d76c51770>
* “Five Steps to Begin Collecting the Value of Your Data” Lean-Data web page: <https://www.lean-data.nl/tag/operational-metadata/>
* “Getting Started with DevOps Automation” by Jared Murrell: <https://github.blog/2020-10-29-getting-started-with-devops-automation/>
* “Incident Management in the Age of DevOps” Atlassian web page: <https://www.atlassian.com/incident-management/devops>
* “An Introduction to Dagster: The Orchestrator for the Full Data Lifecycle” video by Nick Schrock: <https://www.youtube.com/watch?v=MF5OaQEOF2E>
* “Is DevOps Related to DataOps?” by Carol Jang and Jove Kuang: <https://www.dataops.dev/dataops-vs-devops>
* “The Seven Stages of Effective Incident Response” Atlassian web page: <https://www.atlassian.com/incident-management/incident-response>
* “Staying Ahead of Debt and Downtime” by Etai Mizrahi: <https://www.secoda.co/blog/staying-ahead-of-data-debt>
* “What Is Metadata” by Michelle Knight: <https://www.dataversity.net/what-is-metadata/>