rudderstack™

The Data Maturity Guide

A practical approach to customer data infrastructure

Get on the path to data maturity

The Modern Data Stack began as a simple, efficient stack, but it's grown into a sprawling ecosystem of tools. It can be hard to know where to start or what a practical implementation of these tools should look like.

What's needed is a real-world roadmap to help you progressively build a more mature data function. That's why we developed the data maturity journey – a simple four-stage framework that cuts through the chaos to help you architect a practical customer data stack for every phase of your company's journey to data maturity.

Following the steps outlined in these pages will help you to Identify where you are on your data maturity journey, optimize your existing stack, and build your infrastructure with best practices as you grow.

Are you ready to take a more practical approach to your customer data infrastrucure?

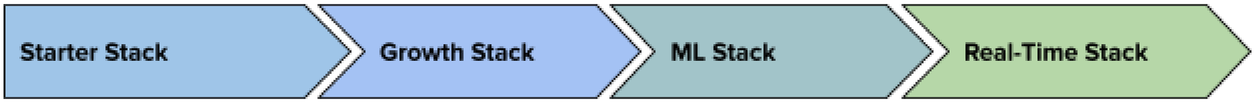
Let's get started.

Table of contents

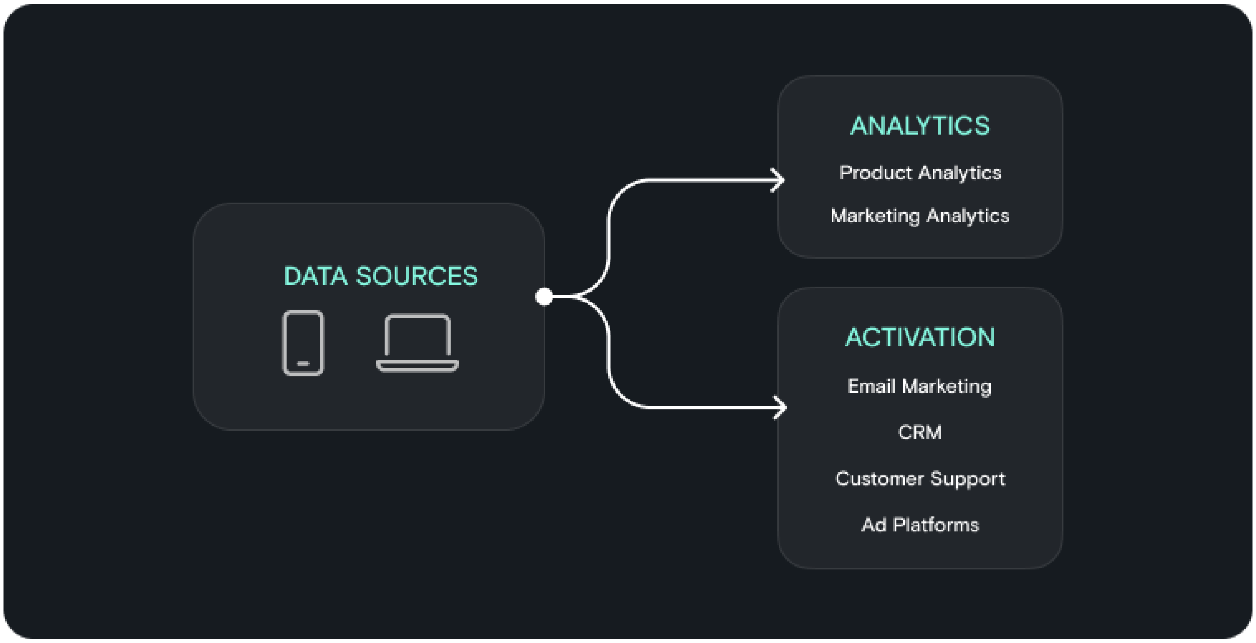
The Four Phases of The Data Maturity Journey

To kick things off, let's properly introduce you to the Data Maturity Journey. It's a simple framework to help you identify the right data infrastructure components for your company based on complexity of needs, level of data sophistication, maturity of existing solutions, and budget. Our framework focuses on customer data and frames data stacks along four phases:

- Starter
- Growth
- Machine Learning
- Real-Time



Phase 1: Starter Stack

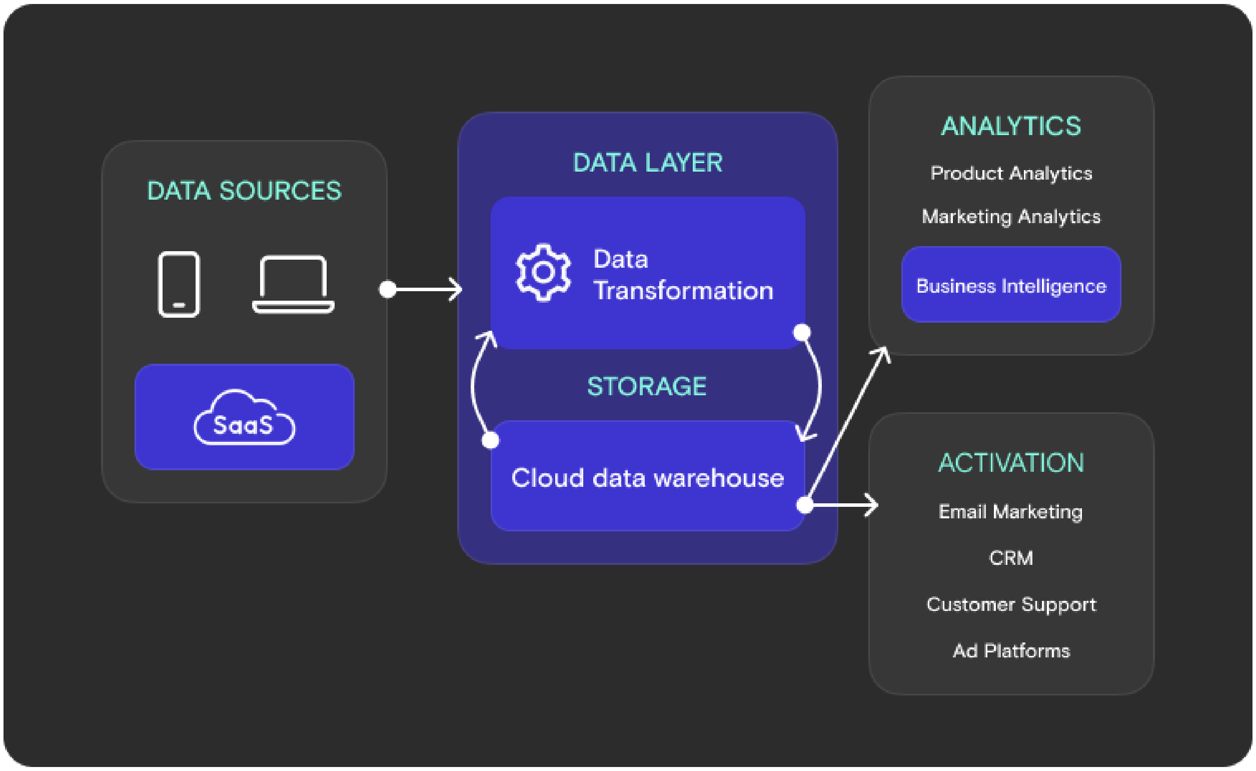


In this phase, you're collecting data from your websites and apps and sending it to multiple downstream destinations. You likely face a few common challenges:

- Different teams want data delivered to their preferred applications
- Brittle data integrations create drain on the engineering team
- Multiple SDKs slow website and app performance

In this phase, introducing a unified data layer will significantly reduce engineering bottlenecks.

Phase 2: Growth Stack

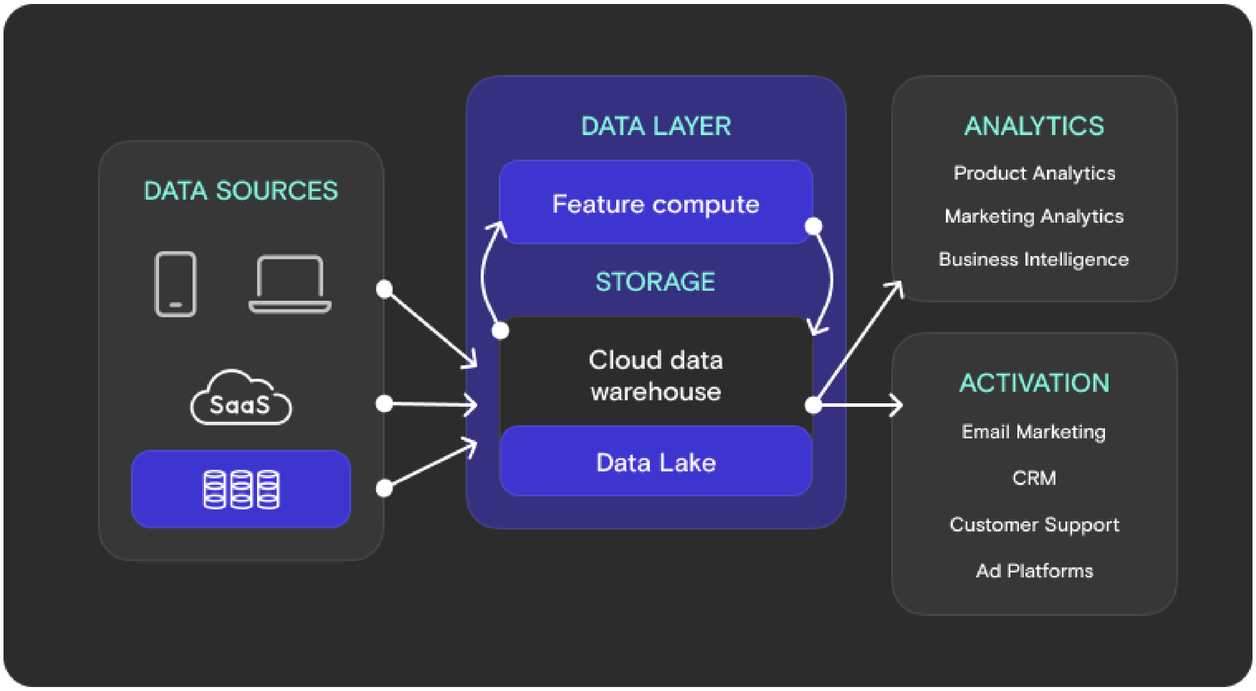


As your company grows, the proliferation of data destinations causes data silos and leads to a number of new challenges:

- Useful data is now trapped in data silos
- Better business intelligence and more personalized customer activation requires data transformation in the warehouse
- Data transformed in the warehouse (e.g., customer cohorts) needs to be delivered to downstream destinations

In this phase, you'll begin centralizing data into a warehouse which will become the single source of truth for analytics. You'll begin making use of transformations in the warehouse and sending the insights into downstream destinations.

Phase 3: Machine Learning Stack



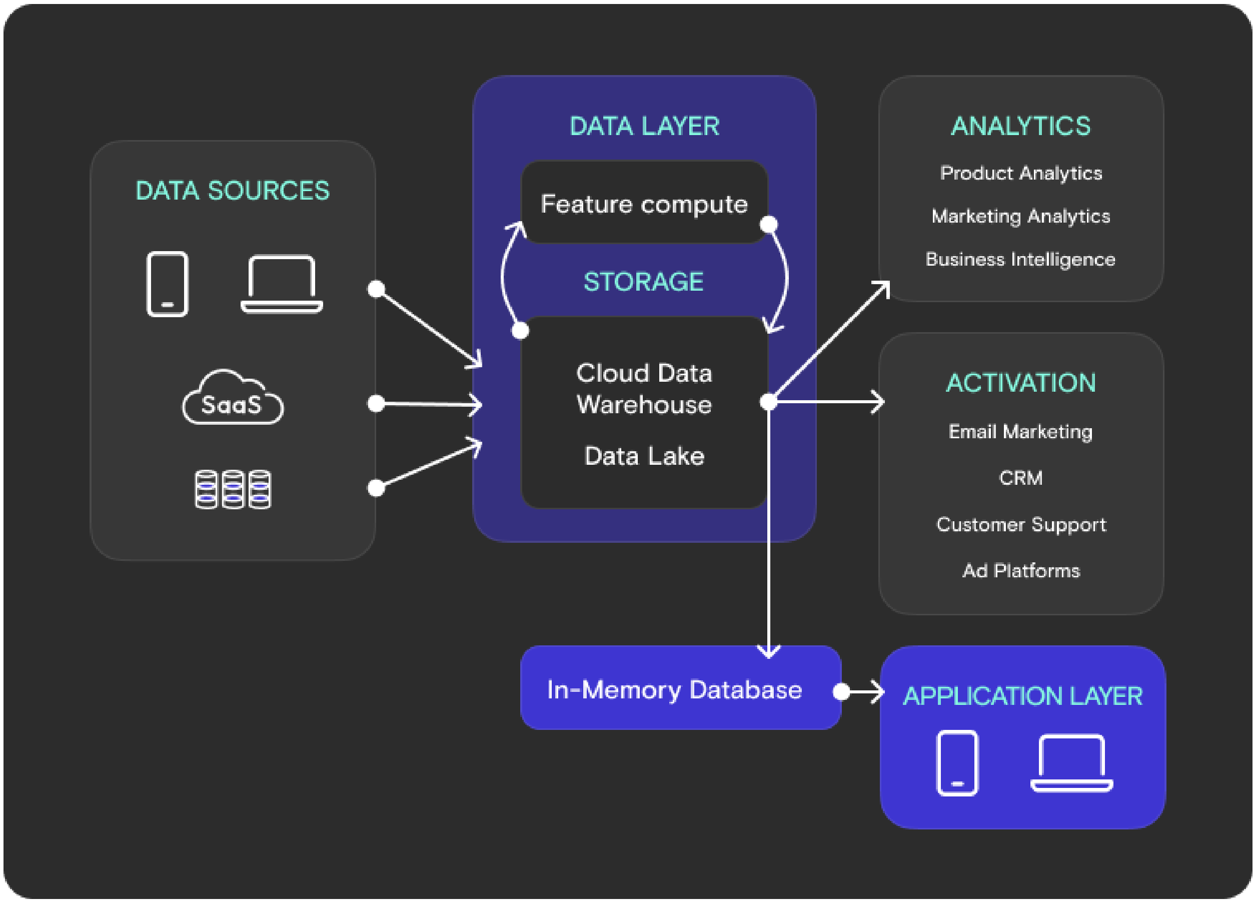
Once the warehouse fulfills your organization's need to answer business questions and serve as a guide for historical data, you'll face another new challenge:

- To drive further optimization, you need to move beyond historical analysis into the world of predictive analytics

Predictive analytics allow you to predict expected user behavior based on early signals and optimize marketing activities accordingly.

In this phase, you'll introduce a data lake along with a machine learning toolset. Investing in the Machine Learning Stack allows you to leverage valuable unsctructured data and mitigate problems proactively, as opposed to reacting after they have been observed.

Phase 4: Real-Time Stack



After moving from historical analysis into the world of predictive analytics, there's one final frontier to address another challenge:

- With your current stack, it's impossible to deliver personalized experiences in real-time

While not every company will reach this stage, for the one's who do it's critically important. Here, you'll introduce an in-memory database to begin serving insights back into your application layer in real-time.

Chapter One

First Step: Starter Stack

If you work in data, you're familiar with "the modern data stack". You also are probably aware that the term is confusing and definitions of what a modern data stack is - along with the related architectural diagrams - are all over the place.

More importantly, the modern data stack is too often positioned as a single bold step for a data team, as in, "Let's implement the modern data stack!"

Data practitioners know that the reality on the ground is different. Data stacks grow and change along with the needs of the business. For data teams doing the work, the goal is actually to progressively *modernize* your data stack in practical ways that impact the businesses ability to make better decisions with data. It's not about adopting a particular architecture just because it is considered modern.

Modernizing a stack is a process that takes time and thoughtful consideration at each juncture—and there are many steps to take as your business grows and changes. The good news is, you don't have to take a single, overwhelming step. You can start making small changes wherever you are in your journey, to better meet the data needs of your business.

Begin your journey

Every company is on a journey to mature their data stack, whether it's an eCommerce company that needs better analytics or a Fortune 500 company trying to implement machine learning projects.

The Starter Stack is the first phase of The Data Maturity Journey. It's where most companies set up their first data stack or take their first step in modernizing their existing infrastructure–be that a conglomeration of Google Tag Manager and 3rd-party scripts, basic ETL jobs, or exporting and uploading CSVs.

It's important to note that the starter designation doesn't refer to company stage (i.e., it's not just for startups). As we will outline below, this phase of the journey is for companies of any size who are facing data integration and data consistency problems due to the lack of a unified data layer.

Starter Stack overview

When it comes to investing in a data stack, **the first step is the most important**. The foundational decisions you make about your data stack now have a big impact on the future. It's critical to get this first step right because the robust, clean, and scalable data layer of the starter stack makes it easier to progress to later phases, like the Growth Stack and the Machine Learning Stack, where you'll build advanced analytics and enrichment as well as predictive models.

The Starter Stack introduces a **unified data layer** that addresses two fundamental data challenges that companies must solve first in order to enable more powerful data use cases:

- **Data Consistency:** Using multiple systems to create and manage customer data caused inconsistency across tools in the stack
- **Integration Complexity:** Every company faces data integration challenges, whether they are a small company using a few SaaS tools or an enterprise with multiple legacy systems

Consider this scenario: The user record in your marketing email system and your app's database are not in sync, so you're unable to confidently answer the question "What is user abc123's current email address?". Intercom says X, the app says Y. When did they last log in? Was it today? Was it two months ago?

As any data professional knows, if data consistency and data integration aren't solved at the root with a unified data layer, it doesn't matter what kind of advanced technology you use in other parts of the stack—other tools will have fundamental limitations because they are only as good as the data you feed them.

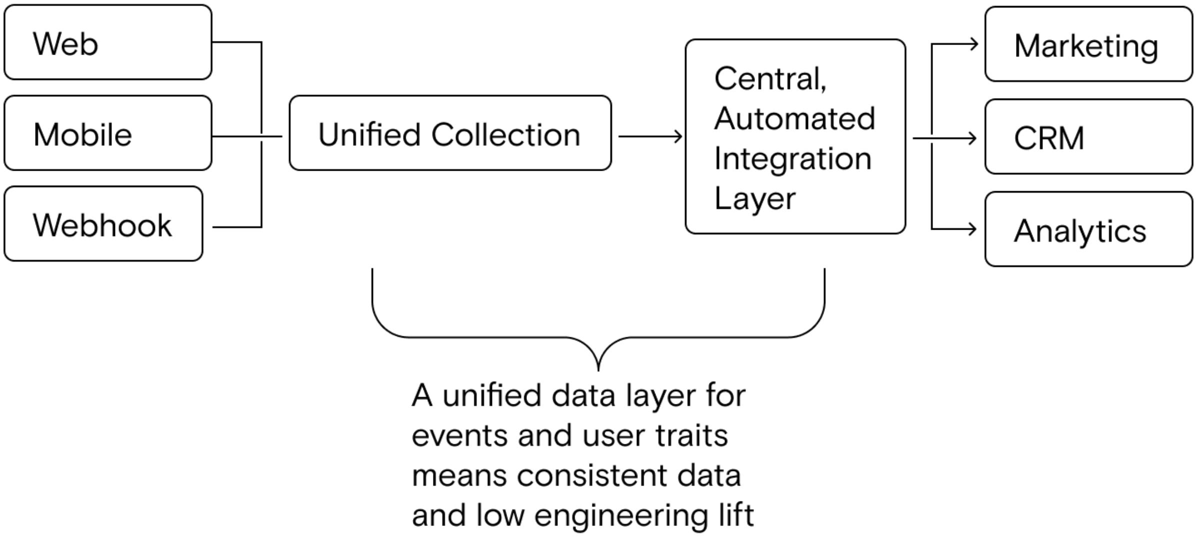
Without a unified data layer, it's also really painful to add new tools to your stack and efficiently maintain the system because you have to take on a completely new instrumentation project.

The Starter Stack solves these problems by:

- Creating a single source of truth for your customer data. This leverages a single system that automatically creates and updates both customer behavioral data and customer traits, which combine to make up user profiles
- Removing the need for point-to-point or custom integration work by unifying all integration needs into a single, automated integration layer. This moves you from a spider web of tool connections to a hub and spoke model, where you push and pull from tools to a central truth store, eliminating any worry about how well tool A's native integration with tool B supports your use case

These problems are related, but distinct. Data can become inconsistent both at the source and in the process of flowing through integrations, so it is critical to separate these concerns in the architectural solution.

Here's what Starter Stack architecture looks like at a high level:


Web
Mobile
Webhook
Unified Collection
Central, Automated Integration Layer
Marketing
CRM
Analytics
A unified data layer for events and user traits means consistent data and low engineering lift

When is it time to implement the Starter Stack?

A key aspect of finding success on your data maturity journey is knowing what steps to take, and when. Taking a step too late (or too early) can result in squandered time, wasted money, and pointless effort. It's important to determine the right time to set up your Starter Stack.

Also, if this seems like a big, daunting project on its face, don't worry. We've talked with thousands of data engineers, and implementation of a basic starter stack like the one above can be implemented quickly—anywhere from a week to a few months, depending on the complexity of your data and number of integrations. Take that into consideration when you think about the amount of time you're paying employees to spend on manually moving data around.

The symptoms

If you're experiencing constant data-related pain, decreases in velocity due to data consistency issues, or data integration problems, you're probably in need of a unified data layer.

The specific nature of these pain points vary, but here are a few symptoms:

- Point-to-point integrations between SaaS tools require constant maintenance related to custom fields, data types, and inability to customize syncing
- The “same data” is different across different tools, meaning different teams have different versions of the truth
- Simple scorecard metrics for your business like visitors, leads, conversions for the last week, month, or Y2D require a lot of manual work to create ad-hoc
- You’re unable to answer complex questions about your customer’s journey, without a herculean manual effort
- You’ve tried to solve some of these problems by applying engineering hours, but you’re quickly learning that software engineering and data engineering are two different skill sets and forcing the former to do the latter is not sustainable

What your company and team might look like

As we said before, companies of any size can begin their data maturity journey by implementing the Starter Stack—it all depends on your specific needs and challenges. That said, Direct-to-Consumer companies often feel the pain earlier than B2B companies, even though both are solving the same basic set of problems.

If your company is smaller in size, you’re likely:

- Using developer time to tackle data engineering and data integration work, which is often a significant distraction from building the core product, your site, or tending to other important matters
- Appointing someone to perform an ops role. Whether it’s official or on an ad hoc basis, you have someone from marketing or sales ops getting into the nitty gritty of integrations and data munging to solve data problems

If your company is larger, you’re likely:

- Building a dedicated data or analytics team
- Running a custom web/app platform that relies on multiple legacy tools and custom data integrations - all of which require maintenance
- Slow to respond to data and integration requests from outside teams because of the complexity and interrelatedness of the app/site and integrations codebase
- Having to regularly export data from back-end systems to combine with data pulled from the SaaS tools that your Analysts and other teams use

- Facing the challenge of modifying external tools, which is costly in both engineering time as well as QA and code deployments, all of which are risky

Let’s look at an example...

To make this crystal clear, let’s examine a sample company that’s ramping up and ready for a Starter Stack.

The company

You’re an eCommerce company, large or small (company size doesn’t matter). Your website, mobile, and marketing teams focus on driving digital purchases through your site and app. You also have a Sales team supporting wholesale buyers. Many of your sales prospects are long-time repeat digital purchasers who would benefit from opening an account.

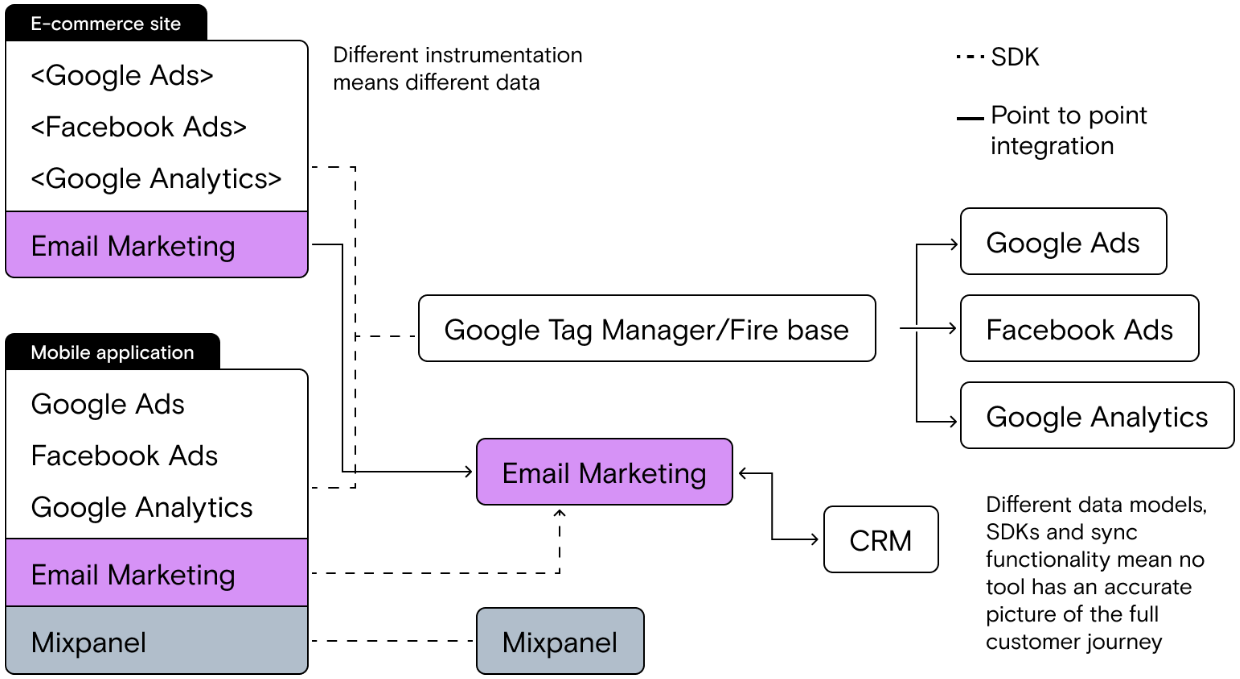
The data stack

Your website runs on a 3rd-party platform, like Shopify or BigCommerce. On the site, your Marketing team runs scripts for Google Ads, Facebook Ads, and Google Analytics, all via Google Tag Manager. Their email marketing tool is connected to the eCommerce platform with a point-to-point integration.

Similarly, your mobile app also runs scripts for Google Ads, Facebook Ads, and Google Analytics via Google Tag Manager and Firebase. Plus your Mobile team runs the Mixpanel SDK for mobile product analytics. As the mobile app is custom, you also run the script from the email marketing tool directly in the app.

The Sales team uses a CRM that gets its data from a point-to-point integration with the email marketing tool.

Here's what the stack architecture looks like:



Your data challenges

With a stack like this, there's no question that you're facing data challenges. Here are a few specific pain points:

- Despite multiple overhauls of the Google Tag Manager implementation, conversion data across Google Ads, Facebook Ads and Google Analytics is inconsistent, making it hard for your Marketing team to know what's working. What's worse is it's hard to triangulate this data across platforms for both web and mobile
- The eComm platform's traffic data is different from Google Analytics data, making it tricky to triangulate actual websites, measure mobile visit volume from various sources, and get clarity on first-touch attribution. Further complications arise from discrepancies between Google Analytics and Amplitude for mobile data. Ultimately, there is no single source of truth for your analytics
- Browser privacy settings and ad-blocking extensions are exacerbating the discrepancies between your systems
- Your email marketing tool has much more flexible data fields than your CRM, meaning only a subset of data can be shared across the tools. This limits your Sales team's ability to see the full customer record in their tool of choice. This also means that your eComm platform, email marketing tool, and CRM all have different versions of the 'same' customer

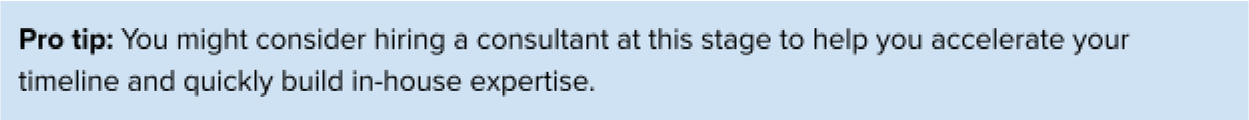
record. In fact, the team using the CRM often gets customer records manually via an ad-hoc contact form

- Your Web and Mobile engineers are constantly bombarded with tickets related to Google Tag Manager and conversion instrumentation. The troubleshooting data discrepancies keep rolling in
    - It's important to note that data instrumentation is not their job / area of expertise. That means whenever they do get to working on those tickets it will take them a lot longer than it would for dedicated data folks who live & breathe event stream data / analytics
- Someone on your team, likely an Ops or Admin person spends too much precious time fiddling with the point-to-point integrations across the eComm platform, email marketing tool, and CRM. They're spinning their wheels fielding requests for data from the Marketing and Sales teams and fixing issues when data doesn't look right in existing tools
- Without a single source of truth, even your primary data wrangler doesn't know what data to trust
- These challenges demand that numerous team leads expend valuable staff and resources for countless hours per week exporting data, running complicated vLookups or macros in a spreadsheet, and manually updating charts
- Despite all of the laborious efforts you still only have basic reporting, and no one really trusts their reports

The Starter Stack playbook: Implementing a unified data layer

So, how do you address and overcome all of the problems above? With a unified data layer, of course!

But first, let's talk about what your goals should be for the Starter Stack and, more importantly, what your data focuses should be in this phase.



Pro tip: You might consider hiring a consultant at this stage to help you accelerate your timeline and quickly build in-house expertise.

Goals of the Starter Stack

The Starter Stack will make it 100x easier to get accurate analytics in your SaaS analytics tools and to better understand your customer, but the reality is that deriving insights about your customer is a never-ending journey that generates new requirements as your business grows and evolves. Getting the foundational data right is the first step.

The goals for the Starter Stack stage are simple:

- Create data consistency by unifying data collection at the source
- Stop wasting time on low-level integrations dev work

If you get those two things right you'll solve your biggest pain points and create a foundation that makes it easy to deliver increasingly complex data products as your company grows.

Data focuses of the Starter Stack

Once you begin modernizing your stack and solving the problems outlined above, you're going to get energized about all of the things you could potentially fix or modify.

It's incredibly important, though, to make sure you get the data fundamentals right.

When you do, every subsequent step becomes easier and you can avoid costly technical debt and re-work.

In this stage, two main types of data serve as the foundation:

- Behavioral data
- User traits

These behaviors and traits come from your websites and apps as users interact with them. These data points are often called events, and this category of data is commonly called event stream data, clickstream data, or user behavioral data.

User traits are most often collected when a visitor makes the transition from anonymous to known-user by providing you with some identifying information, such as an email address or info provided through the creation of an account on your storefront.

If you review the challenges mentioned above, they all relate to:

- **Analytics** on User actions
- **Customer Records** of User characteristics

Specifically, purchases, ad platform conversions and website pageviews are all behavioral events, while customer traits are sent from your website or app to populate customer profiles in the email marketing tool, CRM and eComm platform.

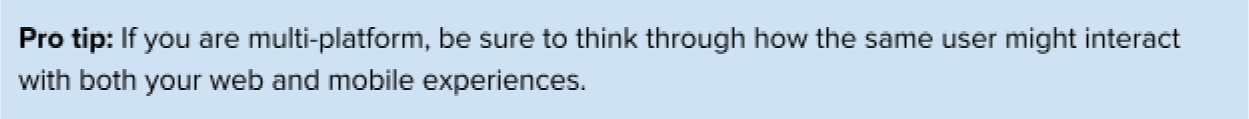
At this stage, keeping things simple is key.

Until you have consistent data on your users and what they do, adding more data sources and data types into the equation will only complicate things.

Step one: Map your user journey

One of the main culprits behind data inconsistency is that implementing tracking of user actions often happens ad-hoc for a specific tactic or occasion. For example, your Marketing team may want to track subscriptions to their new Daily Deals email. This seems simple enough in isolation, but in reality it's actually part of a complex customer journey.

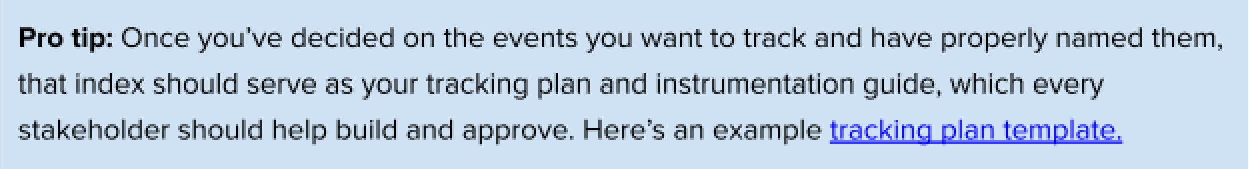
Without considering the bigger picture, it's easy to instrument one-offs that create technical debt later. The way teams name these ad-hoc actions can often create confusion in data because they aren't thinking about the names of all of the other actions being tracked (i.e., what do newsletter, newsletter_1 and subscriber mean?).



Pro tip: If you are multi-platform, be sure to think through how the same user might interact with both your web and mobile experiences.

Before you start implementing the Starter Stack, it's key to map out your customer journey. From their first page view all the way through to the actions they might take after becoming a customer. You may not need to track every single action from the outset, but the context will help you make better decisions about:

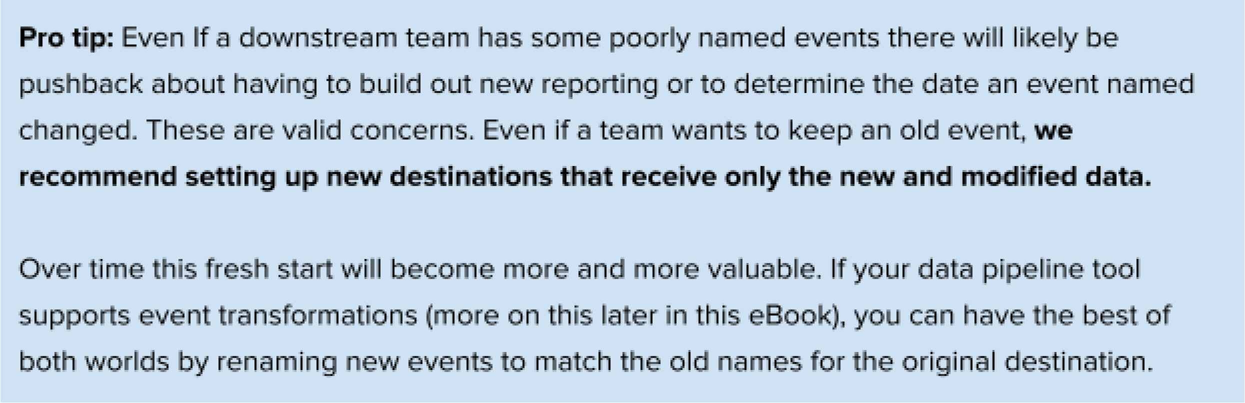
- What you need to track
- The naming taxonomy you use to describe the actions



Pro tip: Once you've decided on the events you want to track and have properly named them, that index should serve as your tracking plan and instrumentation guide, which every stakeholder should help build and approve. Here's an example tracking plan template.

Mapping your tracking plan against current actions and conversions

Now that you've decided what you want to track and how you want to name events, you should map your plan against the User actions and conversions currently being sent to your downstream platforms. This allows you to see which events are net-new as opposed to those which already exist (and/or need to be modified to match the tracking plan). It will also give every stakeholder visibility into what's going to change.



Pro tip: Even If a downstream team has some poorly named events there will likely be pushback about having to build out new reporting or to determine the date an event named changed. These are valid concerns. Even if a team wants to keep an old event, **we recommend setting up new destinations that receive only the new and modified data.**

Over time this fresh start will become more and more valuable. If your data pipeline tool supports event transformations (more on this later in this eBook), you can have the best of both worlds by renaming new events to match the old names for the original destination.

Step two: Create consistency at the source

Inconsistent data from the source happens for one very simple reason: You're sending the 'same' data point, using different code, to each separate platform (which all have their own APIs).

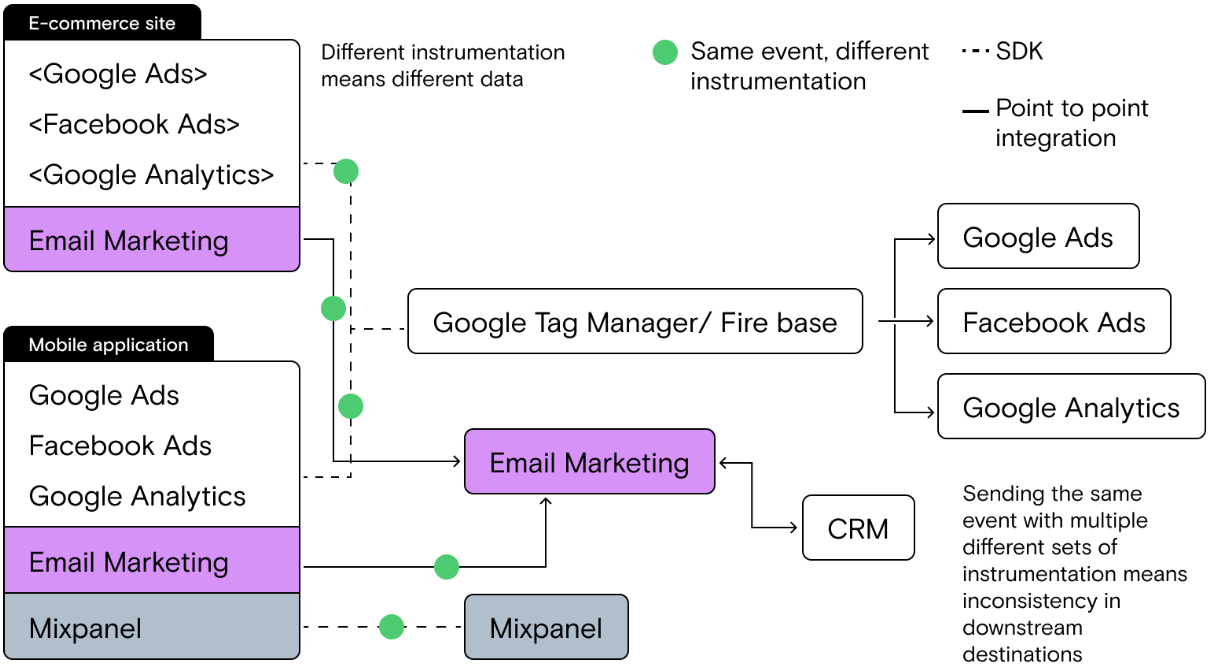
For example, Facebook's code is **fbq('track', 'AddToCart');** vs Google's code, which is **gtag('event','add to cart',{});**.

Let's look at a simple User action like an "Add to Cart" event...

Looking at our pre-Starter Stack architecture, that single event is tracked four (4) different times on the website:

- One event for Google Ads
- One event for Facebook Ads
- One event for Google Analytics
- One event from the eComm platform (sent to the email marketing tool)

On mobile, the addition of Mixpanel means the event is sent five (5) different times. This is show in the diagram below:


E-commerce site
<Google Ads>
<Facebook Ads>
<Google Analytics>
Email Marketing
Different instrumentation means different data
Same event, different instrumentation
SDK
Point to point integration
Mobile application
Google Ads
Facebook Ads
Google Analytics
Email Marketing
Mixpanel
Google Tag Manager/ Fire base
Google Ads
Facebook Ads
Google Analytics
Email Marketing
CRM
Mixpanel
Sending the same event with multiple different sets of instrumentation means inconsistency in downstream destinations

Anyone who has worked in a stack like this knows that the potential problems are endless: Script sequencing, changes in event names (or bad naming in general), various methods of deduplication in the destination platforms and, of course, any sort of error in instrumentation.

Consistency in tracking user actions/events

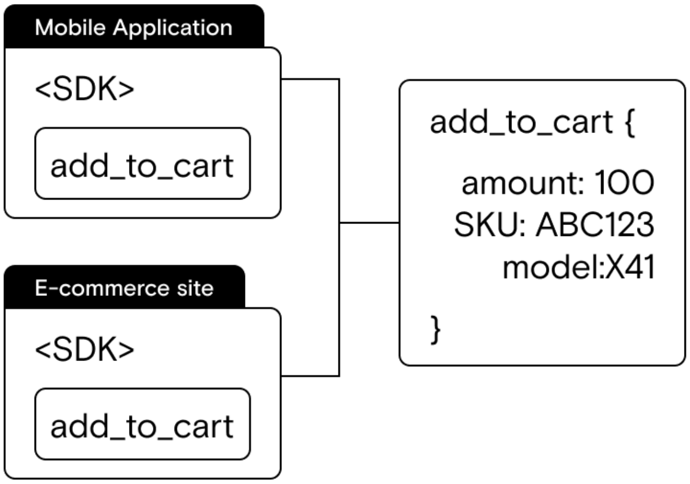
The Starter Stack solves this event fragmentation problem by:

- Abstracting event collection away from the downstream destinations
- Using a single SDK to collect events and identify users
- Using a consistent format for all events and users (this is most commonly achieved using JSON payloads)

If we return to our "Add to Cart" event, the result in the Starter Stack is a single event in a standardized format, as opposed to multiple events in different formats:

```
{
  "type": "track",
  "event": "add_to_cart",
  "properties": {
    "amount": 100,
    "SKU" : "ABC123",
    "model" : "xyz"
  }
}
```

Instead of instrumenting multiple SDK and multiple versions of the same event, we instrument one SDK and one consistent version of the event in every platform, like



One SDK, one event format, CONSISTENT DATA!

Pro tip: While it may be tempting to consider building your own SDK and event format, it rarely works out in the long run. Why? Because you're simply trading integration work for SDK work that never ends. We highly recommend using a data tool that provides SDKs and standardized event schemas right out of the box. This will allow your team to focus on the data, not on building SDKs.

Consistency in tracking user traits

Driving consistency for user traits operates the same way as tracking user actions, the difference is that you are identifying a user.

When a user becomes known (by making a purchase, subscribing to a newsletter, etc.), identifying the user in a consistent format, with the same data, ensures that there's only one version of the 'customer record'. **Here's an example of how you might identify a user:**

```
{
  "type": "identify",
  "traits": {
    "name": "Name Surname",
    "email": "sample@example.com"
  },
  "userId": "hashed_user_id"
}
```

Step 3: Decrease dev work with an integrations layer

Now that you've solved for data consistency at the source, the goal of a unified data layer is to ensure all of the downstream tools have the same copy of events and User records. At face value, these single-format payloads are problematic because the APIs for downstream destinations are unique—that's why you had to run multiple scripts from multiple platforms in the first place.

That's where the integrations layer comes in.

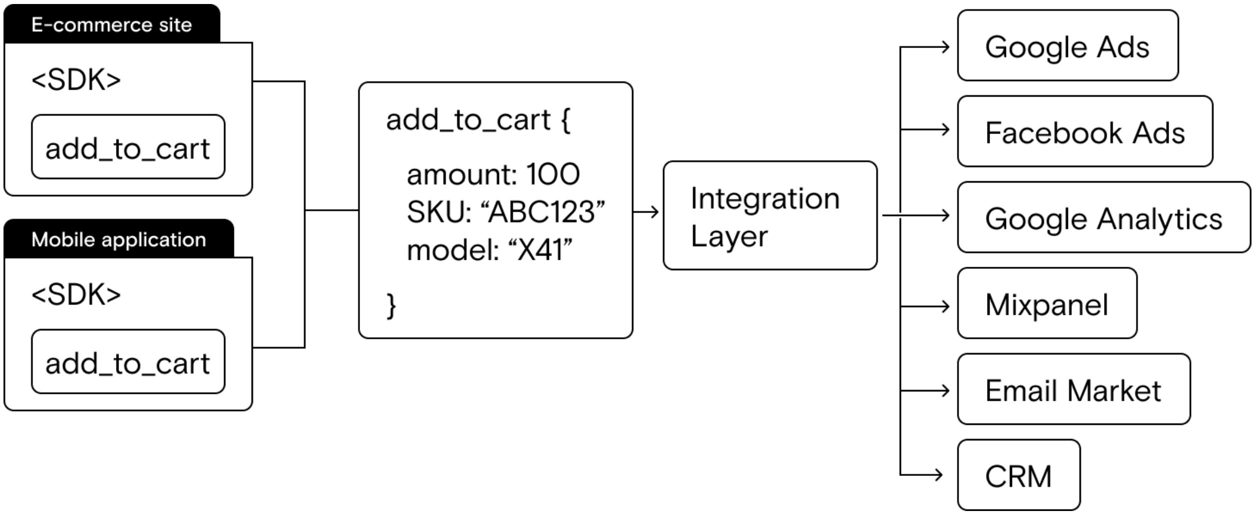
Instead of running multiple SDKs and sending events in various formats, you translate the *standardized* payloads for each destination, so each different API receives the same set of data.

What do we mean by Integrations Layer?

The integrations layer is a post-processing layer, after your standardized payload comes into the platform. An integration layer should allow you to modify or customize a payload on a destination-by-destination basis.

This can take various forms, ranging from product features of CDPs to custom ETL processing tools, such as Apache Airflow or Pipedream, that receive your standard payload before it reaches its destination.

Pro tip: While standardization makes integrations easier, recommend using a third-party data tool that provides an integrations layer. APIs change all of the time, so even with standardized payloads, maintenance debt for homegrown integrations can become significant over time.



Bonus: Consistency and data-fixing with event transformations

Even if you capture events consistently at the source, there will inevitably be problems. People aren’t perfect, so instrumentation can be accidentally modified in releases, creating misnamed data points in your payloads and causing problems in downstream destinations.

On the other hand, downstream teams are known for changing things as well. Let’s say your marketing team renames a field on the customer record in their email marketing tool and, as a result, your identification payloads won’t update that field anymore.

An event transformation layer helps you avoid these issues by allowing you to modify payloads in-flight, on a per-destination basis, without having to go back and deploy code fixes.

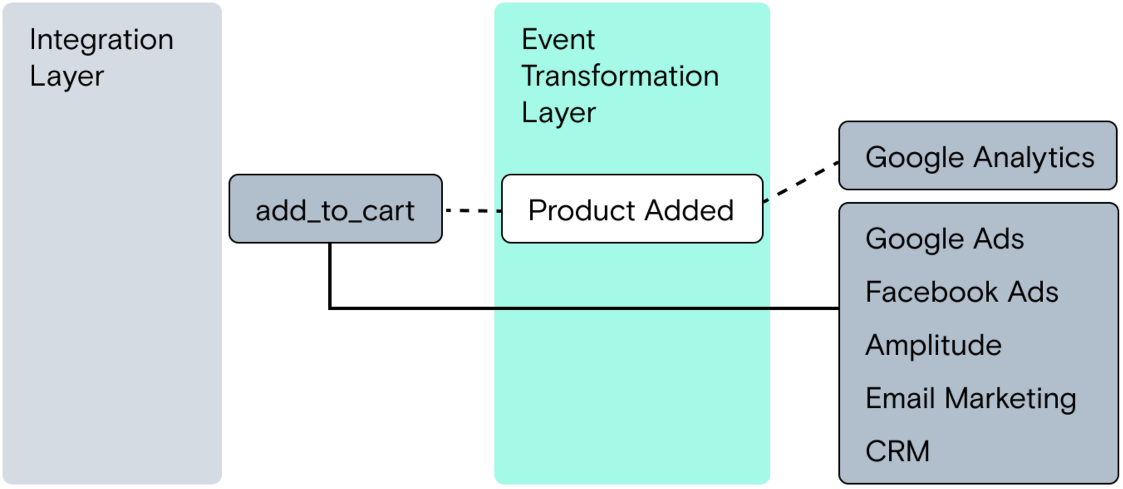
So, if your Sales team changes a field name in their CRM, you can modify identification payloads to automatically map the correct trait to the updated field name. This means engineers don’t have to modify instrumentation and wait on web or app releases. Plus, fixes are applied only on destinations that need them. The original payload remains unchanged at the source and is delivered, as usual, to destinations that don’t need transformations applied.

Most importantly, event transformations allow downstream teams to have their cake and eat it too: You can send events with the new taxonomy to fresh analytics setups (i.e. a new Google Analytics property and Mixpanel project), but keep the old destinations and simply transform the new event names to match the old ones.

This makes transition periods for analytics possible without any interruption in reporting. It also allows you to test new tools with little investment and little risk. Just make the project, hook up the new destination, and you're sending the same data over without changing any code!

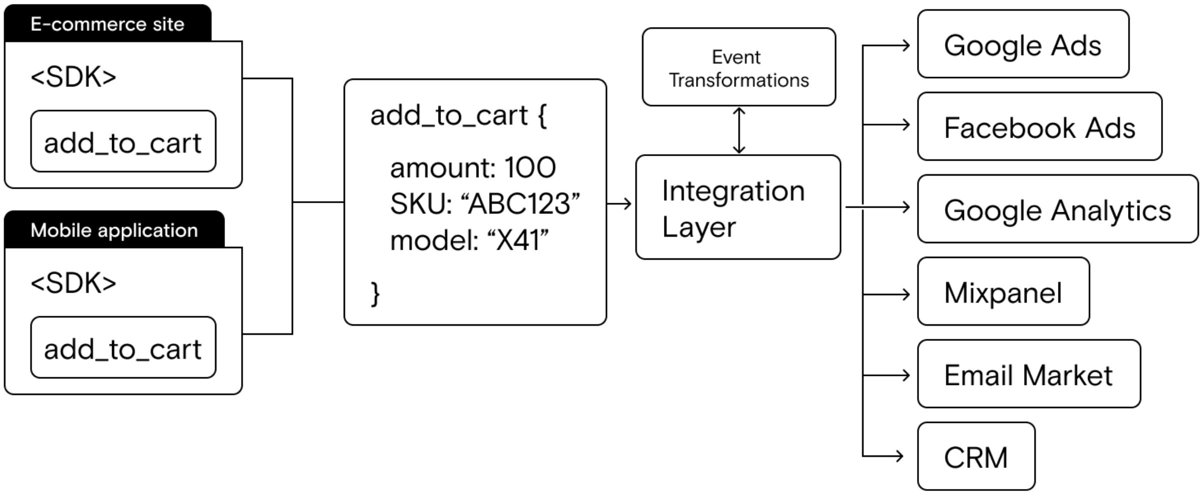
Event transformations help solve point-to-point integration problems where there are differences in things like field names and field data types.

Here's what the Event Transformation layer looks like:



Bringing it all together

Let's look at the architecture of the complete Starter Stack. You have single SDKs sending the same event to an integrations layer, which translates the payload to match the specific needs of each downstream destinations in your stack.



Tooling and technical Review

Though it’s critically important, the Starter Stack doesn’t require a lot of componentry. Here are the technical pieces you need to implement the Starter Stack:

- One SDK to replace all downstream platform SDKs
- Standardized event format (event schema)
- Standardized user identification format (user schema)
- Integrations layer
- Optional but recommended: Event transformations functionality

The right tools for the job

There are plenty of great data tools out there, so how do you decide which ones to use to implement the Starter Stack? Here are some guidelines:

- Open platforms, as opposed to black-box 3rd-party platforms, will give you deeper visibility into how the tool manages your data
- Flexibility should be king. Even though you want to keep it simple in this stage, your stack will grow in complexity over time. Avoid myopia here. You need a tool that has the flexibility to handle more complex use cases, more technical data tools, and the inevitable list of additional SaaS integrations downstream teams will start using over time
- Developer-focused features will become increasingly important as you move from the Starter Stack to the Growth Stack and customer data infrastructure increasingly becomes the responsibility of a data team. Specifically, the ability to integrate that infrastructure into your development workflows and orchestration will make managing a complex stack easier in the future. Accounting for this now will help you avoid a painful tool migration later on

For flexibility and scalability, look for these features:

- Robust integrations with core data tools (warehouses, data lakes, in-memory stores, streaming pipelines)
- A large library of SaaS integrations
- Ability to stream events from SaaS applications (like email events from your engagement platform)
- Ability to transform event payloads in-flight

- Webhook sources and destinations for semi-custom integrations
- The ability to architect semi-custom integrations via custom sources and / or destinations that allow you to run your own integration code inside the platform without having to manage additional infrastructure yourself
- Dev workflow integrations that allow you to:
    - Manage your setup in a config file (as opposed to just a UI)
    - Manage components of the stack, like event transformations or tracking plans, in version-controlled repos
    - Access platform features via API

Outcomes of implementing the Starter Stack

What can you expect after implementing a unified data layer? In short, all of the problems we mentioned above will be solved:

- You'll finally have clean, rich web and mobile analytics data, consistent across platforms
- Your conversions will match across ad platforms and will match the user and purchase numbers in your eComm platform, email marketing and CRM
- Your Analysts will love you because they can focus on uncovering insights, not cleaning data
- Your Ops team will spend time actually optimizing process within, instead of fighting data fires just to keep the basics running
- Your Developers and Data Engineers can finally focus on delivering true value, as opposed to constantly working on low-level data plumbing problems
- You have a robust foundation to build a more sophisticated stack on top of when the time comes

As strange as it may seem, when you look at the outcomes, the Starter Stack is almost always the most fundamentally transformative step on the Data Maturity Journey.

Signs that you're outgrowing the Starter Stack

It's no secret that when teams have easy access to clean, fresh analytics they feel empowered to ask more interesting questions. Insights from data create a loop: Difficult questions require more data to answer (rinse and repeat, as they say). Eventually you'll begin to outgrow your Starter Stack.

On the analytics front, here are a few clear indicators you might be outgrowing the Starter Stack:

- Your SaaS analytics tools are no longer flexible enough to answer your most important questions
- The insights you want require additional data beyond user actions and user traits

Once you hit this point, you’re ready for a data warehouse. Here you can collect all kinds of customer data (in addition to user actions and traits) and perform deeper analysis using SQL or Python.

Once you get a data warehouse (or perhaps even a data lake!), you’re ready to start building on top of your Starter Stack and implementing the Growth Stack.

Chapter Two

Warehouse-First Foundation: Growth Stack

With your Starter Stack set up and a unified data layer giving you accurate visibility into who your customers are and how they interact with you. Leaders and teams can fundamentally understand what's working and what's not and adjust their work accordingly.

But as people learn more about the customer and the business, they start asking more complicated questions. Once your marketing team has a handle on first-touch attribution, they're able to dig into optimization metrics like customer acquisition cost (CAC) and return on ad spend (ROAS) by channel, campaign and even individual ad. At that point, they also start thinking about multi-touch attribution. Once your product team has a handle on the basic user journey, they start wondering about subsets of customers and how they might use your site and app differently. And, of course, the executive team builds an appetite for advanced reporting, craving more detail on financial metrics like revenue, margin, and lifetime value.

Those questions relate to driving growth through advanced optimization, not simply understanding the current state with accurate data. But advanced optimization requires additional data—data that the Starter Stack can't provide.

When you reach this inflection point as a company, you've outgrown the Starter Stack. It's time to augment your architecture with **bi-directional data flows** and a **centralized data storage/compute layer** to unlock the next phase of optimization. In the Data Maturity Framework, we call that architecture the Growth Stack.

Setting the stage

To understand the Growth Stack, we need to remember what the Starter Stack solves for. The Starter Stack eliminates duplicitous data flows with a single, unified data layer. This layer handles capture and integration for the event and user profile data generated by your websites and apps. Once implemented, it drives consistency across all of the SaaS tools used by downstream teams without painful integration work.

Even if the data across your SaaS tools is consistent, though, each system is still only capable of achieving a local maxima in optimization because each SaaS tool is a terminal destination for data in a linear data flow. This means that even though the data matches across systems, they are all data silos.

You could make the argument that silos don't matter if you have consistent data from the source, but that's not the problem. The problem is that each of those SaaS tools themselves *create new kinds of customer data*.

For example, your email marketing tool creates lots of net-new customer data. Data points like email opens and clicks are events that play a big role in the customer journey—but that data stays trapped in the email marketing tool. Similarly, your Sales team creates valuable data in your CRM related to deal cycles, opportunity stages, risk statuses, etc. and that data stays trapped in the CRM.

These data silos make it impossible to both answer more complex questions and to actually *use* every customer data point, both of which are key to unlocking further optimization.

Growth Stack overview

The Growth Stack introduces *bi-directional data flows* and a *centralized data storage/compute layer* to address two fundamental problems caused by data silos:

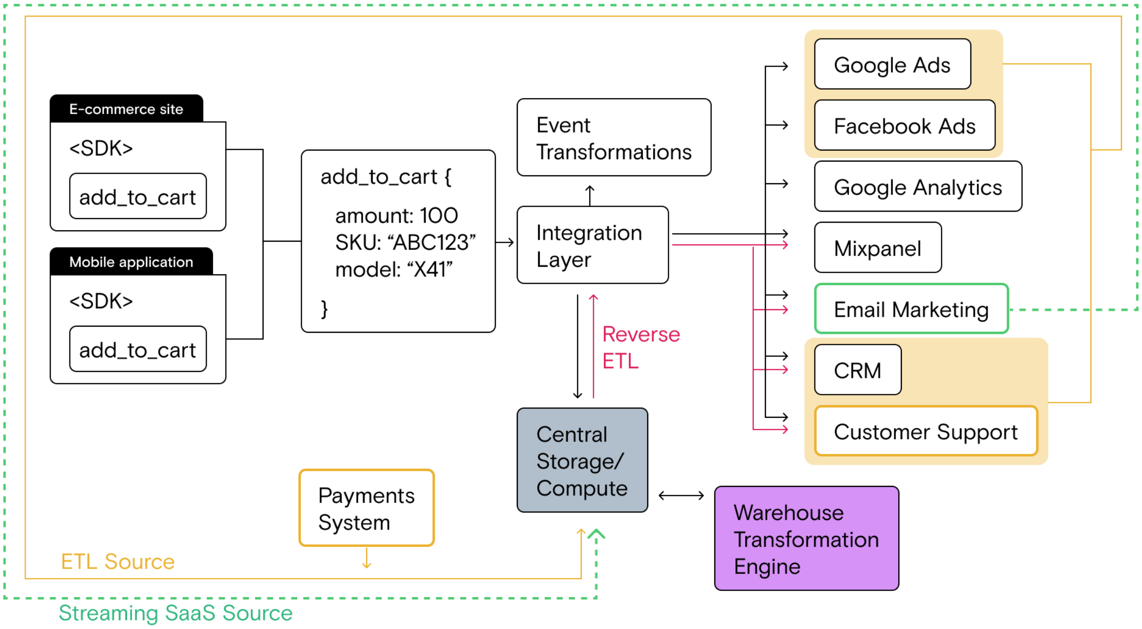
- **Incomplete data:** When data created by downstream tools is trapped in those tools, it's impossible to have a full picture of the customer journey or a customer profile that uses every customer data point
- **Complexity of activating centralized data:** If you centralize data, you get the full picture in one place for analysis, but advanced optimization requires the ability to access every customer data point to *use* in downstream tools

The Growth Stack solves these problems by:

- Introducing a *centralized data storage/compute layer* to serve as a comprehensive source of truth for every customer data point across the stack
- Unifying every customer data point in this central storage/compute layer by:
    - Turning data *destinations* into data *sources*, creating *bi-directional data flows*
    - Pulling data from separate systems (like financial systems) into the central storage/compute layer

- Making the central data store accessible to every tool in the stack through data modeling and *reverse ETL* (this is the activation of centralized data)

As you can see, the Growth Stack involves the addition of multiple new pipelines and data flows, so it's a big step forward in terms of the sophistication of your stack. In the implementation guide below we break down each component and the data flows in detail, here's what Growth Stack architecture looks like at a high level:



When is it time to implement the Growth Stack?

Before we get into the details of the architecture, data flows, and implementation process, let's step back and talk about when you need to implement the Growth Stack.

The symptoms

If you have consistent data across your tools, but teams feel like they've hit a ceiling in their ability to understand and optimize the customer journey (and business), you're ready to start planning a Growth Stack implementation.

Here are a few specific symptoms you might have:

- Teams want data from other tools that are hard to get. For example:

    - Marketing is mapping the customer journey and wants to look at email activity alongside web activity to understand the most effective paths to purchase
    - Sales wants notifications of key activities on the website or in the product, so they have the full context before messaging prospects
    - Support wants to know what Account Manager is assigned to the user that just wrote in with an issue
- The reporting requested by both downstream teams and leadership requires data from systems that aren't connected to the customer data stack. For example, leadership may want to understand margin in the context of coupons, but that data lives in a payments system or ERP
- Teams want to build and analyze customer segments that aren't possible to create in SaaS analytics tools because of different data models or an inability to combine different types of data (i.e., event data and relational data)

What your company and team might look like

When to implement the Growth Stack is less about company size and more about data needs. That said, companies who run the full version of this stack tend to be in the mid-market or above, mostly because running this infrastructure means you have a lot of data and dedicated data resources. Here are a few indicators that you're ready for the Growth Stack.

If your company is on the smaller end of the spectrum, you:

- Have brought on your first dedicated data resources, but centralizing data is still a major challenge
- Find downstream teams reverting back to point-to-point integrations to get the data they need (despite your efforts to centralize data integration), re-introducing the data consistency problem
- Have explored expensive, enterprise analytics tools as a solution to your visibility and consistency challenges–At this point, these solutions are overkill and the growth stack will save you thousands of dollars/month!

If your company is on the larger end, you:

- Have an established data team or teams
- Have legacy data stack components and data flows that require tribal knowledge to manage
- Are dealing with increasing complexity related to multiple legacy batch jobs that move data between systems

- Have tried to build a full picture of the customer with enterprise marketing software, to no avail

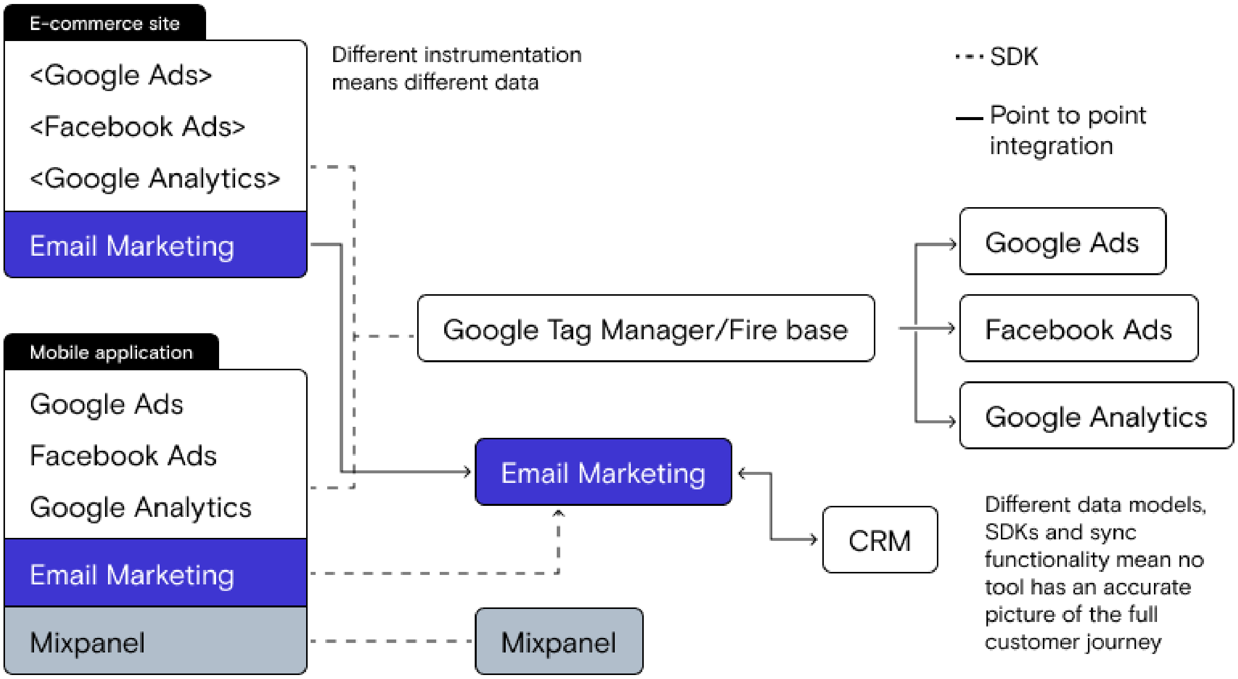
Let's revisit our example company

Remember our real-life example from last chapter? Here's a description of the company:

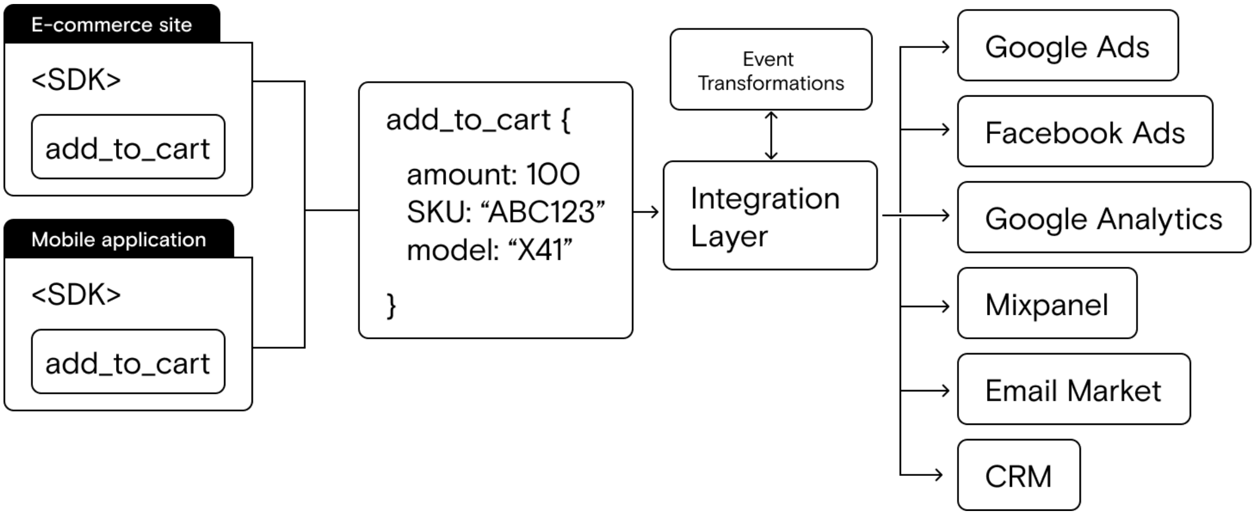
> *You're an eCommerce company, large or small (as we said above, company size doesn't matter). Your website, mobile, and marketing teams focus on driving digital purchases through your site and app, but you also have a sales team supporting wholesale buyers. Many of the sales team's prospects are long-time repeat digital purchasers who would benefit from opening an account.*

By implementing the Starter Stack, you've moved from multiple replicated data flows and inconsistent data to a unified data layer for events and customer profile updates, as shown in the diagrams below.

Before Starter Stack implementation:



After Starter Stack implementation:



The data stack

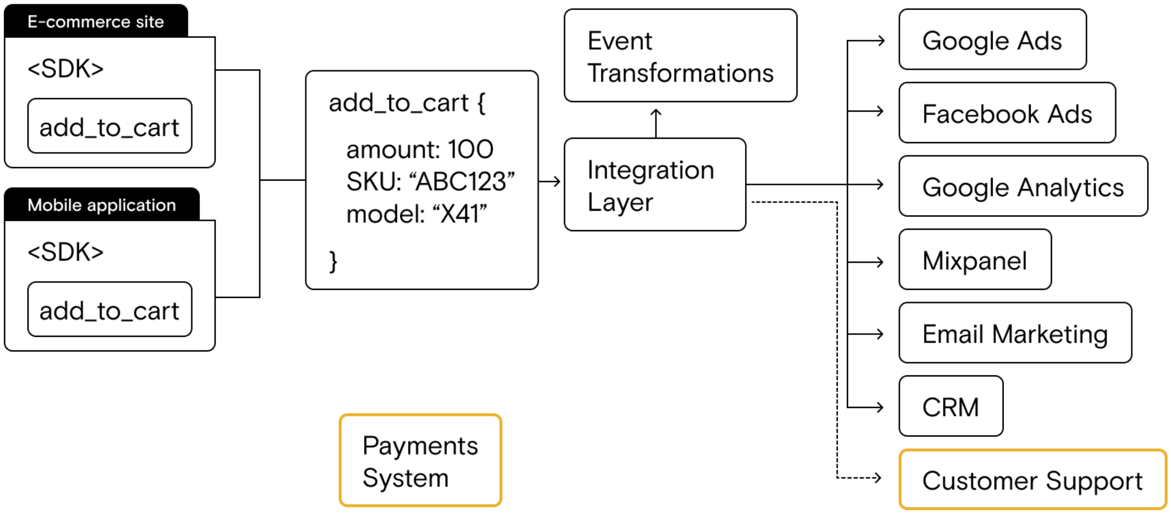
Your Starter Stack streamlined event and user profile updates, so the architecture is really clean: SDKs in your website and mobile app send single user events with a standardized schema through an integrations layer and update these downstream tools with the same data:

- Web analytics
- Product analytics
- Email marketing and automation
- CRM
- Ad platforms

Since implementing the Starter Stack, though, several new tools have entered the picture:

- **Customer support software** – the support team introduced a platform to streamline support for digital purchasers and wholesale customers. Luckily this integrates with your existing data layer for events and customer profile updates
- **Payments infrastructure** – Increased appetite for advanced financial metrics required a payments infrastructure that wasn't fragmented across web and mobile. You implemented a more robust platform, but it creates relational data, not event data, so it doesn't integrate with your existing data layer

Here’s what the stack looks like with the addition of these new tools:



Your data challenges

- Your go-to-market teams have hit the limits of their SaaS tools for web and product analytics. There are two flavors of this problem:
    - Their ability to analyze user behavior across platforms is limited because of the different data formats of their respective analytics tools. For example:
        - Your Marketing team wants to understand the relationship between email and web behavior, but they’re limited to tagging links in emails and using the UTM parameters on pageviews in their web analytics tool. This is messy and doesn’t scale
        - Analysts want to track the full user journey for wholesale customers, but part of the wholesale journey lives offline. Specifically, sales calls are offline events that are recorded in the CRM. These can’t be tracked via event streaming
    - They want to break analytics down by user segments, but the data needed for segmentation is inaccessible because it lives in other tools. For example:
        - The Mobile team wants to better understand how wholesale customers use the mobile app in order to optimize new features for them. The customer type flag, though, lives in Salesforce and is set manually by the

sales team, so it’s not available in the product analytics tool. (And, of course, Dev doesn’t want to touch the “Salesforce custom field to product analytics integration” ticket…yikes!)

- Marketing wants to move from first-touch to multi-touch attribution. They need detailed ad performance data from their ad platforms, but they have no way to join it with user behavior data in their web analytics tool
- Your executive team wants more advanced financial reporting. It’s impossible to produce what they’re looking for in any of your analytics tools, so you’ve resorted back to laborious and error-prone exports and spreadsheets. For example:
  - The Executive team wants to see if different demographic segments use coupons at a higher rate (and thus have lower margins). This is hard to answer because demographic data lives in the CRM and discount data lives in the payments platform
- Your data team (and likely Ops team) is spending time managing integrations again because the demands of downstream teams require ad-hoc and inconsistent liberation of data from silos

The Growth Stack playbook: implementing a warehouse-first customer data platform

The Growth Stack is a response to increased complexity that the Starter Stack can’t handle. Specifically, when use cases arise that require usage of *all* customer data, including new data types, you have to implement new tooling and new data flows.

Goals of the Growth Stack

The Growth Stack enables you to use all of your customer data to build a full picture of your customer. This allows your downstream teams to break through the barriers of localized optimization. They’ll be able to act on insights *and* the data driving them.

There are two goals of the Growth Stack stage:

- Eliminate data silos to collect and centralize all customer data from the stack, giving you a complete view the customer and their journey
- Make that data available for use by every tool in the stack

Data focuses of the Growth Stack

The data focuses of the Starter Stack phase were *behavioral data* (events) and *user traits*. But as SaaS tools and external systems create additional customer data, the Starter Stack can't tell the whole story, and you're creating data that teams can't use. The data focuses of the Growth Stack directly address this limitation.

Growth Stack data focuses include:

- Relational customer data from other systems (SaaS, transactional, etc.)
- *Computed* user traits

Relational data from other systems is simple: CRM data, payments data, etc. This is sometimes called "tabular data" (because it is often in column/row format) or even "structured data" (i.e. a structured data set, as opposed to a single event).

But what do we mean by *computed traits*? If pulling in relational data from other systems is how the Growth Stack accomplishes centralization, computed user traits are the primary way it makes that data available to every other system in the stack. We use the term computed because syndicating data from a centralized store back to downstream tools requires some sort of computation to make the data usable in downstream tools.

Let's say you want to syndicate data points trapped in one system, like your CRM, to another tool, like your product analytics platform. Lead qualification flags are a good example. Your product needs to analyze feature usage by qualification to determine if qualified leads behave differently than unqualified leads, but those qualification flags are set manually by the sales team on user records in the CRM.

In the Growth Stack, you can solve this problem by:

- **Centralization** – Pulling the relational CRM data into your centralized data store
- **Computed traits** – Performing a query to associate CRM qualification flags with user emails/IDs in a user table
- **Activation** – Sending those computed customer traits to the product analytics tool, allowing the product team to analyze behavior for different segments of users

Implementing: A journey in itself

You don't need to implement every component of the Growth Stack at one time. In fact, very few companies do. Instead, you can take a step-by-step approach, starting with the central storage/compute later and building out from there.

While most companies who add a central storage/compute layer will eventually layer in all of the components of the Growth Stack, the K.I.S.S. (“Keep It Super Simple”) principle is key here. Implementing only what you need is a best practice you shouldn’t ignore because unnecessary complexity in your stack is never a good thing.

For example, you’ll want to get a good handle on centralizing data in your warehouse before you add a pipeline to get that data out of the storage layer and back into downstream tools. You’ll avoid a lot of headaches by first solidifying your data models and working to understand which data points downstream teams actually require.

Step 1: set up a cloud data warehouse and feed it with event data

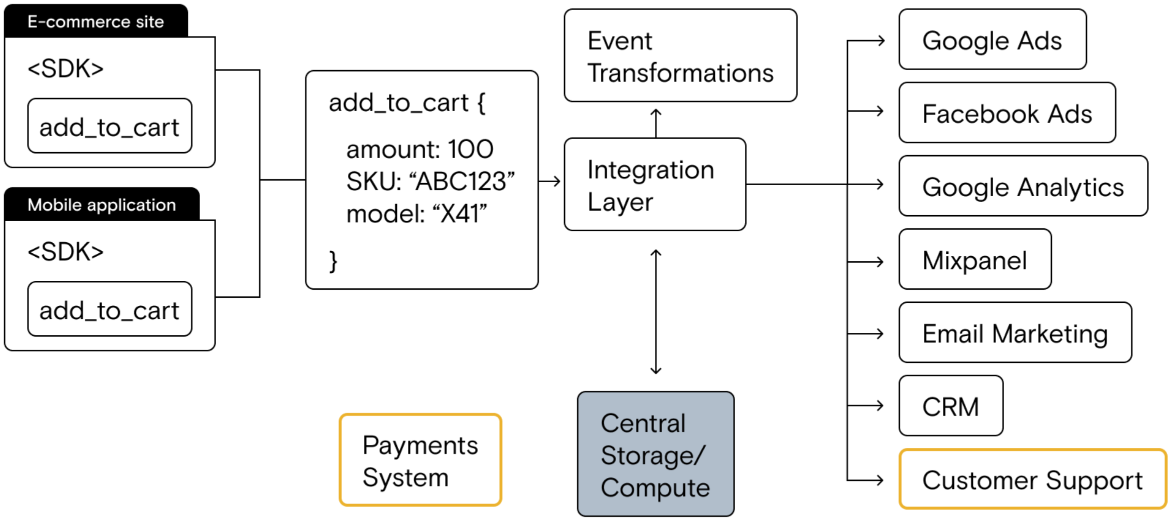
Cloud data warehouse: The core of the growth stack

The *centralized storage/compute* layer is the cornerstone of the Growth Stack. The tool for the job is a cloud data warehouse. It’s the key that unlocks all the benefits of the Growth Stack.

This step is easy, thanks to an array of solid data warehouses that offer self-serve plans to get you started. In fact, you can probably stand up a basic warehouse instance in half an hour.

Feeding your warehouse with your existing event and user profile data

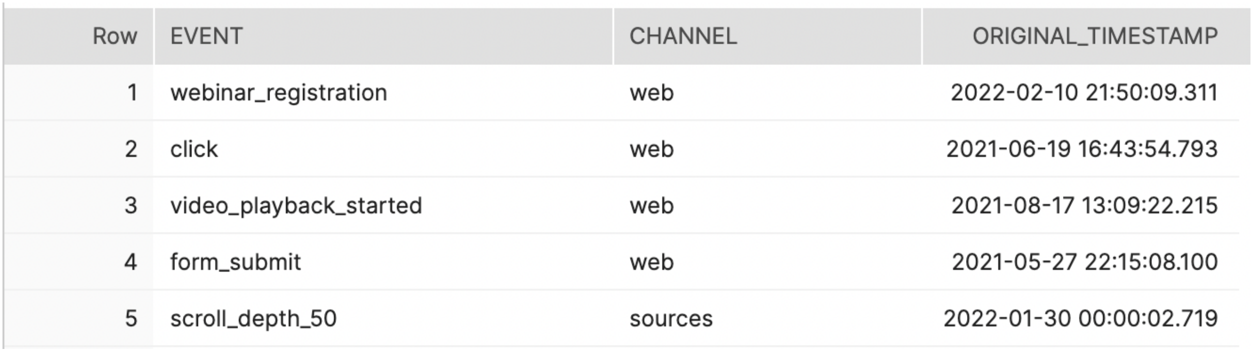
If you’ve made the right choice on the tooling you used to collect and route events and user profile traits in the Starter Stack, you’re in luck. Routing the raw payloads from that feed is now as simple as adding your warehouse as a destination in the integration layer.



Ideally, your event streaming solution automatically flattens the raw payloads into standardized schemas, which show up as a standard set of tables in your data warehouse. Different tools have different ways of structuring event and user profile data in the warehouse, but almost all of them provide at least two basic types of tables:

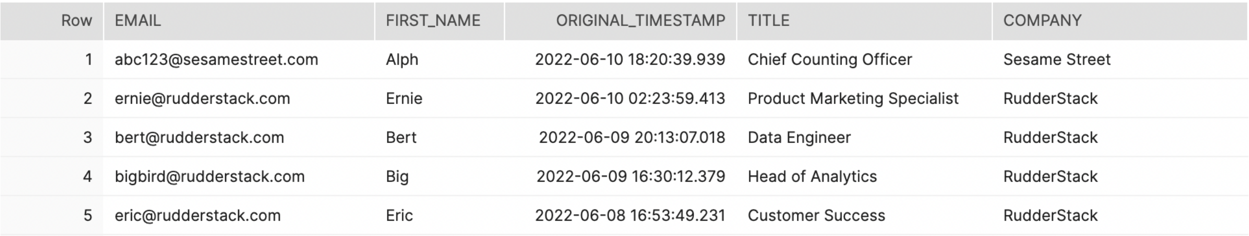
- **User action tables**, in which each row represents a user action (i.e., a single event performed by a user).
    - For example, you'll have a table for pageview actions, a table for click actions, a table for added_to_cart actions, etc.
    - Note that various tools either combine all actions into one table, separate actions into their own tables, or provide both.
- **User profile tables**, in which each row represents a user and the traits that you've associated with that user.

User action table:



Row	EVENT	CHANNEL	ORIGINAL_TIMESTAMP
1	webinar_registration	web	2022-02-10 21:50:09.311
2	click	web	2021-06-19 16:43:54.793
3	video_playback_started	web	2021-08-17 13:09:22.215
4	form_submit	web	2021-05-27 22:15:08.100
5	scroll_depth_50	sources	2022-01-30 00:00:02.719

User profile table:



Row	EMAIL	FIRST_NAME	ORIGINAL_TIMESTAMP	TITLE	COMPANY
1	abc123@sesamestreet.com	Alph	2022-06-10 18:20:39.939	Chief Counting Officer	Sesame Street
2	ernie@rudderstack.com	Ernie	2022-06-10 02:23:59.413	Product Marketing Specialist	RudderStack
3	bert@rudderstack.com	Bert	2022-06-09 20:13:07.018	Data Engineer	RudderStack
4	bigbird@rudderstack.com	Big	2022-06-09 16:30:12.379	Head of Analytics	RudderStack
5	eric@rudderstack.com	Eric	2022-06-08 16:53:49.231	Customer Success	RudderStack

Congratulations! You're now ready to up your analytics game

Getting your raw event and user profile data into the warehouse is a light lift with the right tools, but it makes a huge impact. In the warehouse, you can write SQL to join user actions and profiles in any way you want, leaving the limitations of SaaS analytics behind.

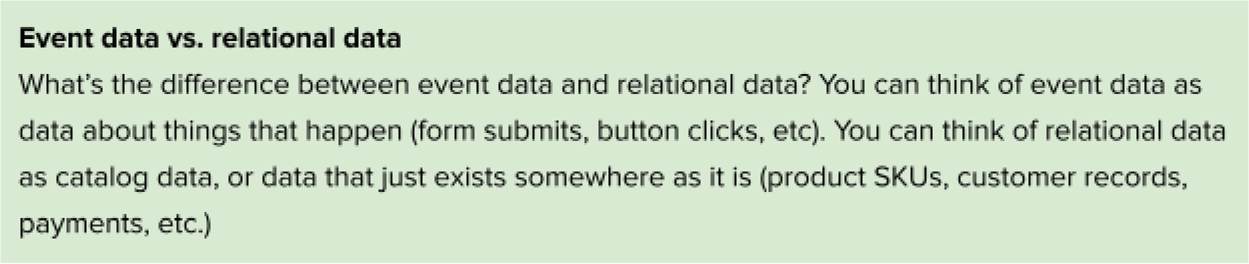
Things get even more exciting when you combine this behavioral data with data that's been siloed in SaaS tools.

Step 2: Use ETL pipelines to eliminate SaaS data silos

This step focuses on pulling in *relational* data from silos across the stack. Because this is a different data type, it requires a different type of pipeline. These pipelines are traditionally referred to as ETL pipelines, which stands for Extract, Transform, Load. Let's break this down:

- You want to *extract* relational data from other systems
- It needs to be *transformed* to match the data format in your warehouse
- Then you need to *load* it into warehouse tables

Most ETL tools load data in batch jobs that run on a schedule. For example, you might load your CRM data every 24 hours to make sure you have the latest information on a daily basis.



Event data vs. relational data

What's the difference between event data and relational data? You can think of event data as data about things that happen (form submits, button clicks, etc). You can think of relational data as catalog data, or data that just exists somewhere as it is (product SKUs, customer records, payments, etc.)

Similar to event data, this relational data is loaded into various tables. Data coming from SaaS tools often loads a table for each type of 'object' in the SaaS tool.

Tables loaded by your CRM might be:

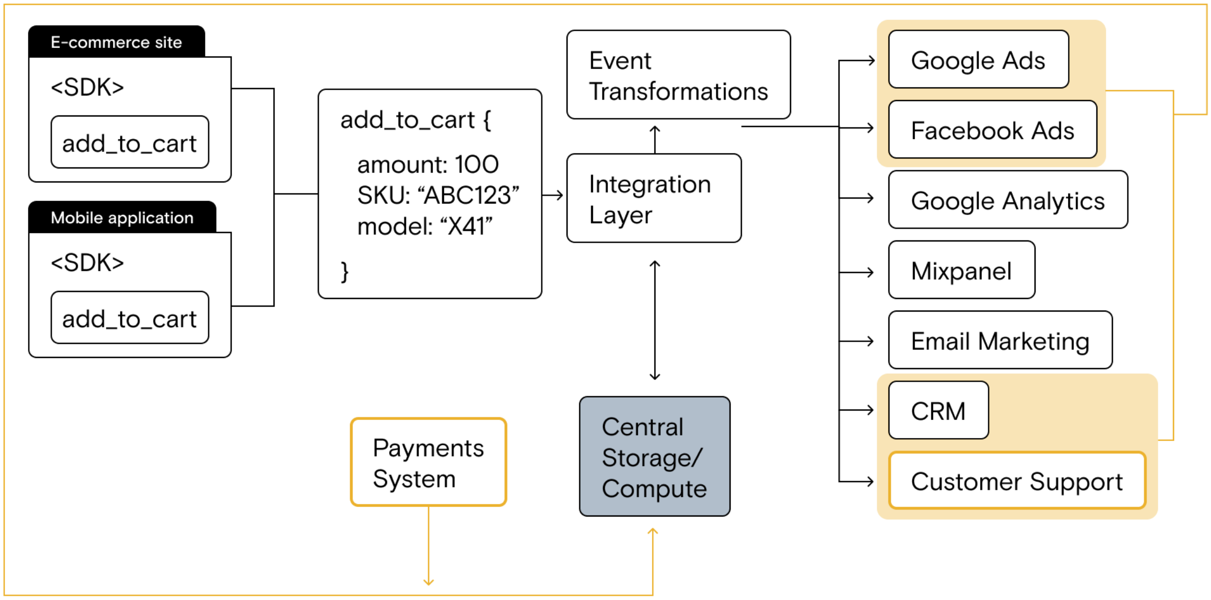
- A **Leads** table for user records

- An **Accounts** table for company records
- An **Opportunities** table for deals created by the sales team

Tables loaded by your Payments System might be:

- An **Orders** table representing purchases
- A **SKU** table representing the product purchased
- A **Customers** table representing the users who made purchases

This is what we call a *bi-directional data flow*: Data flows into SaaS apps, then out of SaaS apps, meaning these apps are both destinations for data and sources of data in the stack.



The power of centralized data

Centralizing your behavioral data *and* relational data in your warehouse is a huge step in modernizing your stack and improving your analytics. By joining various tables, you can answer all kinds of interesting questions. Here's one example drawing from the tables mentioned above:

What are the top pageviews for qualified leads who have made more than 5 purchases in the last year?

With your behavioral, CRM, and payments data in your warehouse, you can join data from:

- The Pageviews table
- The Leads table (which has the qualification flag from sales)

- The Orders table

And with a simple query, you've answered a question in minutes that it would have taken hours or days before!

Step 3: Use streaming sources to pull in event data from SaaS tools

This is perhaps the most neglected step in building out a Growth Stack, but it's one of the most useful data flows. Relational data isn't the only type of data produced by SaaS tools. Lots of SaaS platforms produce *event data*. Email platforms are a great example: email sends, opens, and clicks are all user actions, just like the behavioral data from your website or app. These events are critical for getting a complete view of the customer journey.

If you've chosen your infrastructure well at the Starter Stack phase, your existing system can ingest streaming events from modern SaaS tools. Practically, this means events from your email platform follow the same data flow as events from your website or app. The source is the only difference, one an SDK, and the other the SaaS app itself, which is really nifty.

This is another example of a bi-directional data flow, where the data destination is also a data source.

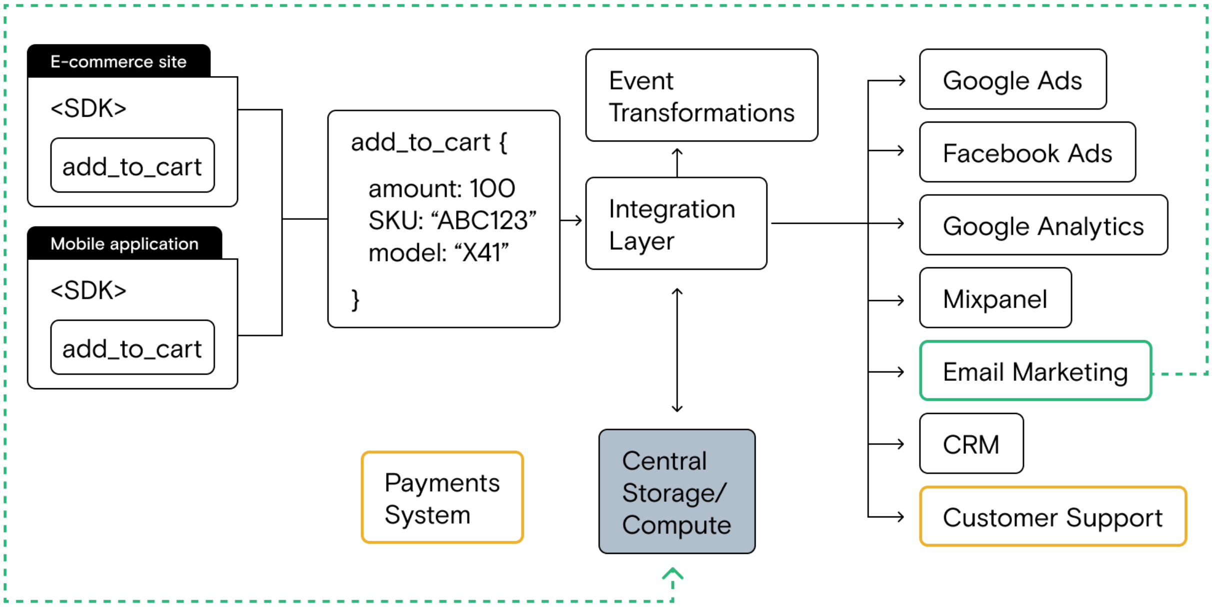
SaaS events in the warehouse

As you would expect, these events are loaded into tables in a similar fashion to other event tables, adding to the list of user actions you can query.

Streaming SaaS events directly to other SaaS tools

One often overlooked data flow that is extremely helpful is what we call *streaming data loops*, where data streaming from one SaaS app is routed through the event streaming infrastructure *directly into another SaaS tool*.

For example, a product team might want to stream email events into their product analytics tool so they can analyze the impact of these events on user onboarding or activation. With a streaming data loop, email opens and clicks can be sent directly to the product analytics tool.



What if there isn't a direct source integration with your event stream?

Inevitably companies run across legacy systems or new tools that don't integrate directly with their event streaming infrastructure. Here, the flexible data layer from the Starter Stack phase pays significant dividends. The most flexible platforms offer Webhook sources that allow you to ingest data sent from other platforms without a direct integration. You'll have even more flexibility if the event streaming infrastructure has a transformations layer that allows you to interact with the payload. With webhook sources, flexible event streaming infrastructure, and transformations, you can do some pretty cool stuff. You can even turn offline events like sales calls from your CRM into actual events in the event stream. When your downstream tools generate events, you can use the events in (almost) real-time, rather than waiting for an ETL sync that may be hours away.

Step 4: Build a complete customer profile and customer journey in your warehouse

At this point you've collected all of your customer data in the warehouse, including:

- Behavioral data and user traits from your websites and apps.
- Relational data from SaaS tools and other systems across the stack.
- Event data produced by SaaS apps.

You're now ready to leverage the *compute* part of the centralized storage/compute layer. In terms of specifics, the sky's the limit because you can query the data in any way you want, but you can start by building complete customer profiles and complete user journeys.

Complete customer profiles - Many people call this "Customer 360" or a "single view of the customer." Whatever the terminology, the concept is simple: You join every customer trait from every tool into a master table of customer profiles.

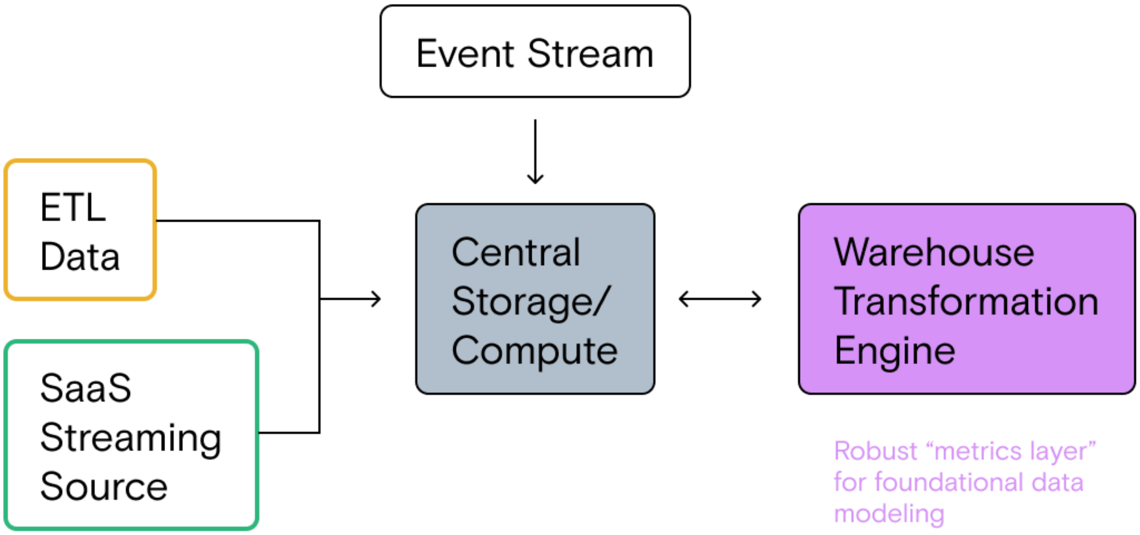
Complete customer journeys - This concept is similar, but instead of traits, you're aggregating every user touchpoint (pageviews, add_to_carts, purchases, email clicks, etc.) with each user. This gives you a detailed map of everything each customer has done.

Insights that can come from querying complete profiles and journeys can be extremely powerful.

Optional: Managing computation with a transformations engine on the warehouse

Even though you have all of the data in your warehouse, building queries for things like a complete customer journey can be non-trivial if you have a lot of data and fragmented user touchpoints.

One way you can mitigate this challenge is to leverage a third-party transformation engine that runs on the warehouse. This makes building, running, and managing core queries easier and more accessible. This tooling is called a *metrics layer* or *analytics engineering layer*, and it enables you to perform more complex transformations on your data than you can with standard SQL.



Robust "metrics layer" for foundational data modeling

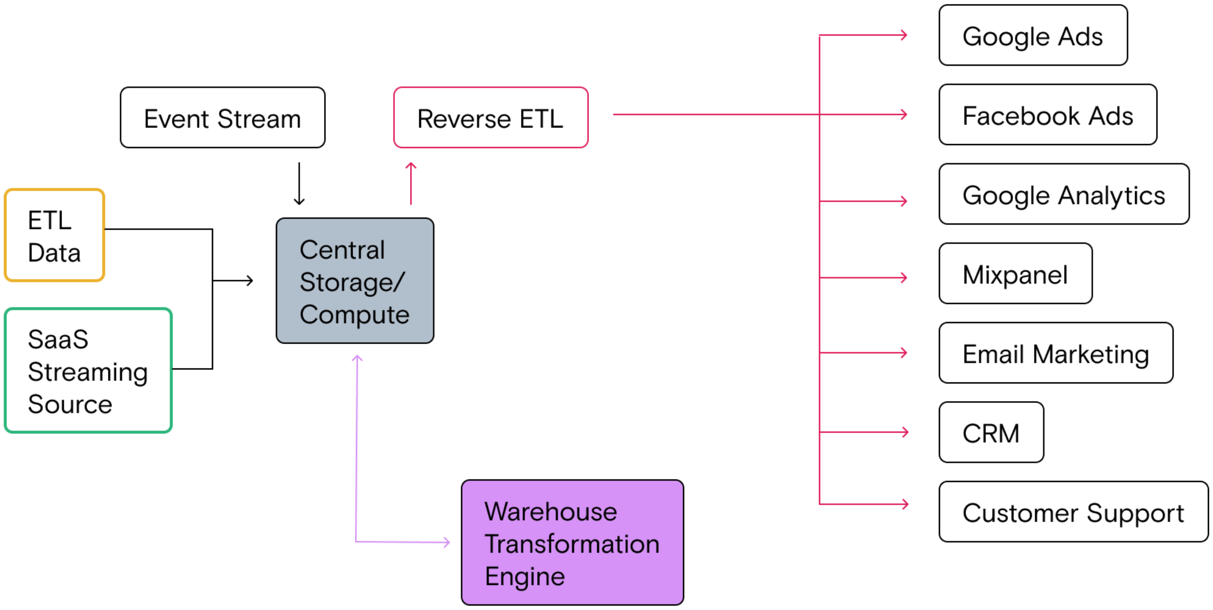
Step 5: Push completed profile data and computed traits out of the warehouse to the rest of your stack

Driving insights with complete data is great, but the biggest win of the Growth Stack is sending that data to downstream tools so teams can act on it. This introduces another new data flow where data is pulled *from the warehouse* and sent to downstream tools.

The pipeline used for this data flow is called a *reverse ETL* pipeline, because it is the inverse of the ETL data flow.

Using reverse ETL pipelines is straightforward:

1. Produce a materialized table in your warehouse that contains the data you want to send to downstream tools
2. Map the data points in the table (i.e., column names) to the fields in downstream platforms
3. Run (or schedule) the sync job



Note that in the Growth Stack, the warehouse itself is bi-directional, serving as both a destination and source.

You can do many interesting things with reverse ETL pipelines, but most companies start with two primary use cases:

1 – Syncing a complete Customer Profile to every tool

Remember the master table of customer profiles we mentioned above? With reverse ETL you can send that master users table to downstream SaaS tools so that every system has the same, comprehensive customer profiles. Said another way, every tool can use every user trait from every other tool. This enables all sorts of powerful segmentation and analytics use cases in downstream tools.

2 – Computing traits and sending them to every tool

Unifying existing traits is great, but the real power comes when you compute traits using various data points in the warehouse. Let's say your company has the following definition for high-value customers:

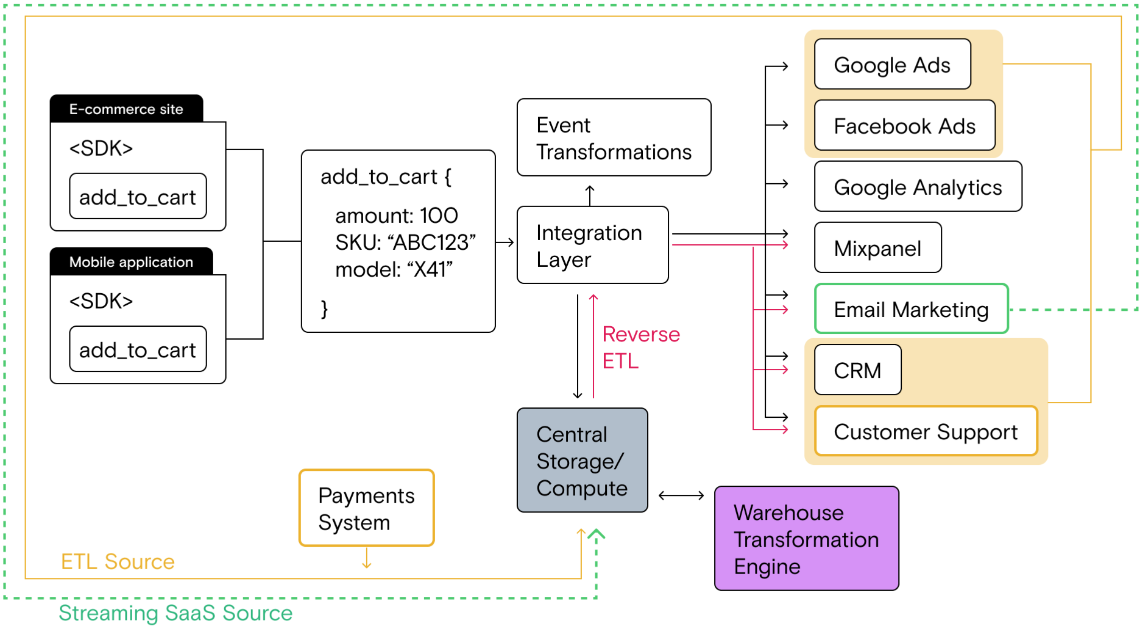
- Have made 5 purchases over $100 within the last year
- Are between 25 and 45 years old
- Visit the website at least once per week

With all of the data in the warehouse, you can easily join purchase data, demographic data from the CRM, and pageview data from the event stream to produce a table of high-value users with a high value flag.

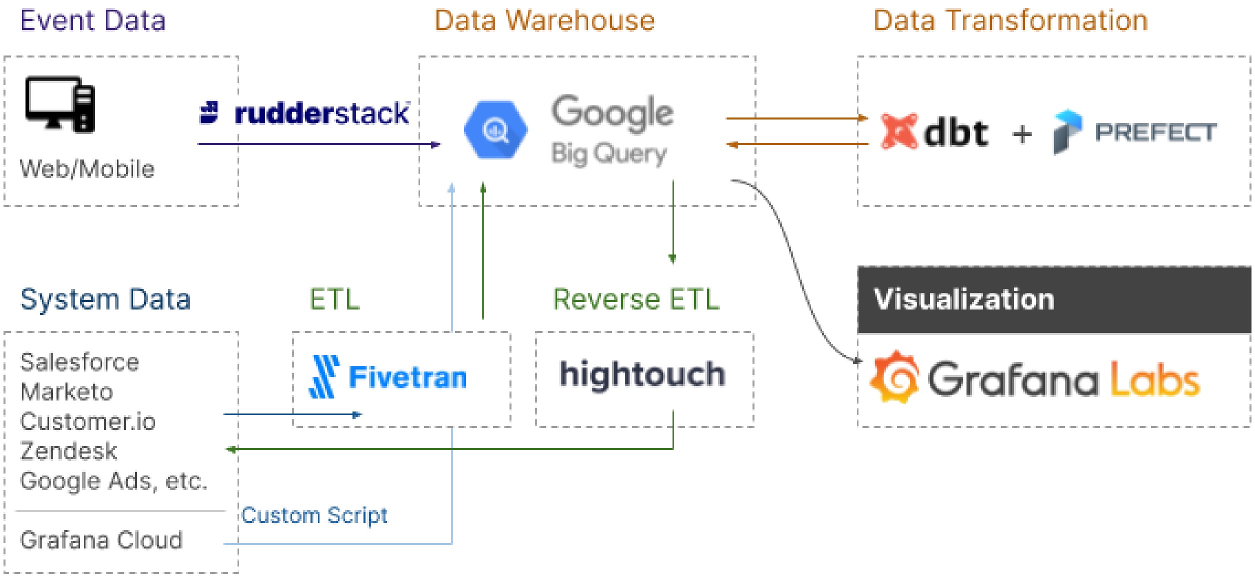
With reverse ETL, you can send that computed *high value* tag from the warehouse to all of your downstream tools. That enables Marketing to easily create dedicated email campaigns and Product to analyze flows and usage for that specific segment in their product analytics tool.

Bringing it all together

Here's a look at the architecture of the full Growth Stack:



To make this even more real, let's look at an example architecture from an actual company. Grafana is an open-source analytics and visualization tool that offers querying, alerting, and more to help companies understand their data. They wrote about the stack they use to drive business intelligence and included an excellent architectural diagram of the Growth Stack.



Tooling and technical review

The Growth Stack adds several key components to the stack:

- A cloud data warehouse
- ETL pipelines
- Reverse ETL pipelines
- Optional but recommended: data transformation engine on the warehouse

The right tools for the job

A few pointers to help you choose the best tools for your Growth Stack.

Cloud Data Warehouse

Most modern warehouse solutions are effective, but here are a few specifics to look for:

- Separation of storage and compute functionality
- Flexible ingestion, which you can often determine by the number of integrations it has with event streaming and ETL data sources
- Scalability and speed. These become increasingly important as your data volume grows

ETL pipelines

ETL pipelines have been around for a long time and there are lots of options, but here are some important characteristics to prioritize:

- A robust library of data sources
- Strong documentation on schemas and load functionality
- Strong scheduling functionality–integration with orchestration tools is a plus to mitigate future complexity
- Error recovery–a good ETL tool will alert you of any errors and resume syncing immediately without requiring manual historic runs to recover data from the downtime
- Optional: Configurable transformation (depending on your specific needs, you might want to transform the data in specific ways before it is loaded into the warehouse)

Reverse ETL pipelines

Data engineers have been sending data out of databases in various ways for a long time, but the emergence of dedicated SaaS reverse ETL pipelines is fairly recent. Here's what to keep an eye out for:

- A robust library of warehouse and database integrations because you'll eventually want to pull from your warehouse and even production databases
- Query functionality to enable you to query from the reverse ETL tool itself
- Strong scheduling functionality (integration with orchestration tools is a plus)
- A Data mapping UI for non-technical users
- Code data mapping functionality enabling technical users to do more complex configuration

Outcomes of implementing the Growth Stack

You don't have to implement every part of the growth stack to drive impact. You get incremental value at each step, and when you do put together the complete stack, the results are significant:

- Eradication of data silos across the stack
- A single, centralized source of truth for every customer data point
- Comprehensive and complete user profiles
- Full visibility into customer journeys
- The ability to perform almost any kind of analytics on any customer data
- The ability to quickly derive valuable computed traits of any kind
- The ability to send data from your warehouse to your entire stack for use in both analytics and business tools

When all of those powers combine:

- Your analysts and downstream teams can focus on deriving insights from analytics, not dealing with incomplete or dirty data sets
- Your downstream teams can leverage any customer data point from anywhere in the stack in their work, making optimization significantly easier
- The data team can spend time partnering with downstream teams to build more interesting data products and use cases, instead of wasting time on low-level integrations work, data cleanup, or manual data syncs
- Perhaps most importantly, your leadership team will have more insight than they've ever had, without the need for data fire drills – executive scorecards that give an instant overview of company performance with standard metrics no longer require any analyst work

Signs that you're outgrowing the Growth Stack

It can can take a long time to outgrow the Growth Stack. In fact, it's not uncommon for a small mid-market company to start with a cloud data warehouse implementation, add additional pieces of their Growth Stack as business requirements demand, and still be running the Growth Stack as a large publicly traded company.

The limitation of the Growth Stack, though, is that it drives *deterministic* analytics and activation. The Growth Stack is the best possible way to optimize your business based on a complete, accurate data-driven view of *what has happened*.

When companies at scale begin to master deterministic analysis, the next phase of growth requires anticipating *what is likely to happen in the future* and acting on it before it does. We call this *predictive* analytics and activation.

If you're at a point where you're beginning to think about predicting customer churn, making personalized recommendations, or customizing your website/app experience for different users, you're going to need to leverage machine learning models and move from the Growth Stack to the Machine Learning stack.

Chapter Three

Getting Predictive: Machine Learning Stack

Before we get to the hot topic of machine learning, let's do a quick recap of where we are in the data maturity journey. In the Starter Stack, you solved the point-to-point integration problem by implementing a unified, event-based integration layer. In the Growth Stack, your data needs became a little more sophisticated—to enable downstream teams (and management) to answer harder questions and act on all of your customer data, you needed to centralize both clickstream data and relational data to build a full picture of the customer and their journey. To solve these challenges you implemented a cloud data warehouse as the single source of truth for all customer data, then used reverse ETL pipelines to activate that data.

At this point, life is good. All of your customer data is collected and sent to your entire stack in real-time to cloud tools via event streaming and in batch loads to your data warehouse (via both event streaming batch loads as well as ETL for relational data). This means you can build a full view of your customer and push complete profiles, computed traits, and even audiences from your warehouse back to your stack via reverse ETL.

Setting the stage

Yes, life is good. But could it be even better? Currently all of the insights and activation your work has enabled are based on historical data—what users have done in the past. The more teams understand about the past, though, the more they think about how to mitigate problems in the future.

Customer churn is a great example. With the right data you can easily see which customers have churned and launch win back emails, but wouldn't it be better to engage those users *before* they churn? This type of magic is the next frontier of optimization, and it requires moving from historical analytics to *predictive* analytics. Predictive analytics enable you to determine the likelihood of future outcomes for users.

Predicting future behavior and acting on it can be extremely valuable for a business. Consider the example above. When you leverage behavioral and transactional data signals to identify customers who are likely to churn and proactively incentivize their next interaction, you're more likely to prevent them from churning. The same is true for new users: if you can predict their potential lifetime value, you can customize offers accordingly.

This is a big step for your data team and your data stack because it introduces the need for predictive modeling, and predictive modeling requires additional tooling. The good news is that if your data is in order, which it should be if you implemented the Growth Stack, you already have a running start.

Machine Learning, ML Ops, and keeping it simple

Depending on a number of factors you are considering in your analysis, and the types of data you're using, the most scalable way to answer predictive questions is to use machine learning. Machine learning is a subset of artificial intelligence at the junction of data engineering and computer science that aims to make predictions through the use of statistical methods.

It's worth noting from the outset that not all predictive problems have to be solved with formal machine learning, though. A deterministic model (i.e., if a customer fits these characteristics, tag them as being likely to churn) or multivariate linear regression analysis can do the job in many cases (hat tip to SQL and IF statements!). Said another way, machine learning isn't a magic wand you can point at any problem to conjure up a game-changing answer. In fact, we would say that intentionally starting out with basic analyses and models is the best first step into predictive analytics—another good reminder of the K.I.S.S. principle when building out your data stack and workflows.

If you are familiar with machine learning, you know that there is also an entire engineering discipline focused on the tools and workflows required to do the work (this is often called ML ops or ML engineering). ML engineering is different from data engineering, but the two disciplines must operate in tandem because **the heart of good predictive analytics is good data**. This is why many data scientists play the role of part-time data engineer.

Taking the first step

One other important thing to note: Machine Learning is a wide and deep field. The most advanced companies build fully custom tooling to support their various data science workflows, but for many companies complicated data science workflows are overkill. The good news is that with modern tooling, you don't have to implement a massive amount of ML-focused infrastructure to start realizing the value of predictive modeling.

To that end, this isn't a chapter about ML ops, workflows or various types of modeling. Our focus will be showing you how many companies take their *first steps* into predictive analytics by making simple additions to their toolset and leveraging existing infrastructure (the Growth Stack) to operationalize results from models.

So, without further ado, let's dive into the stack itself.

Machine Learning Stack overview

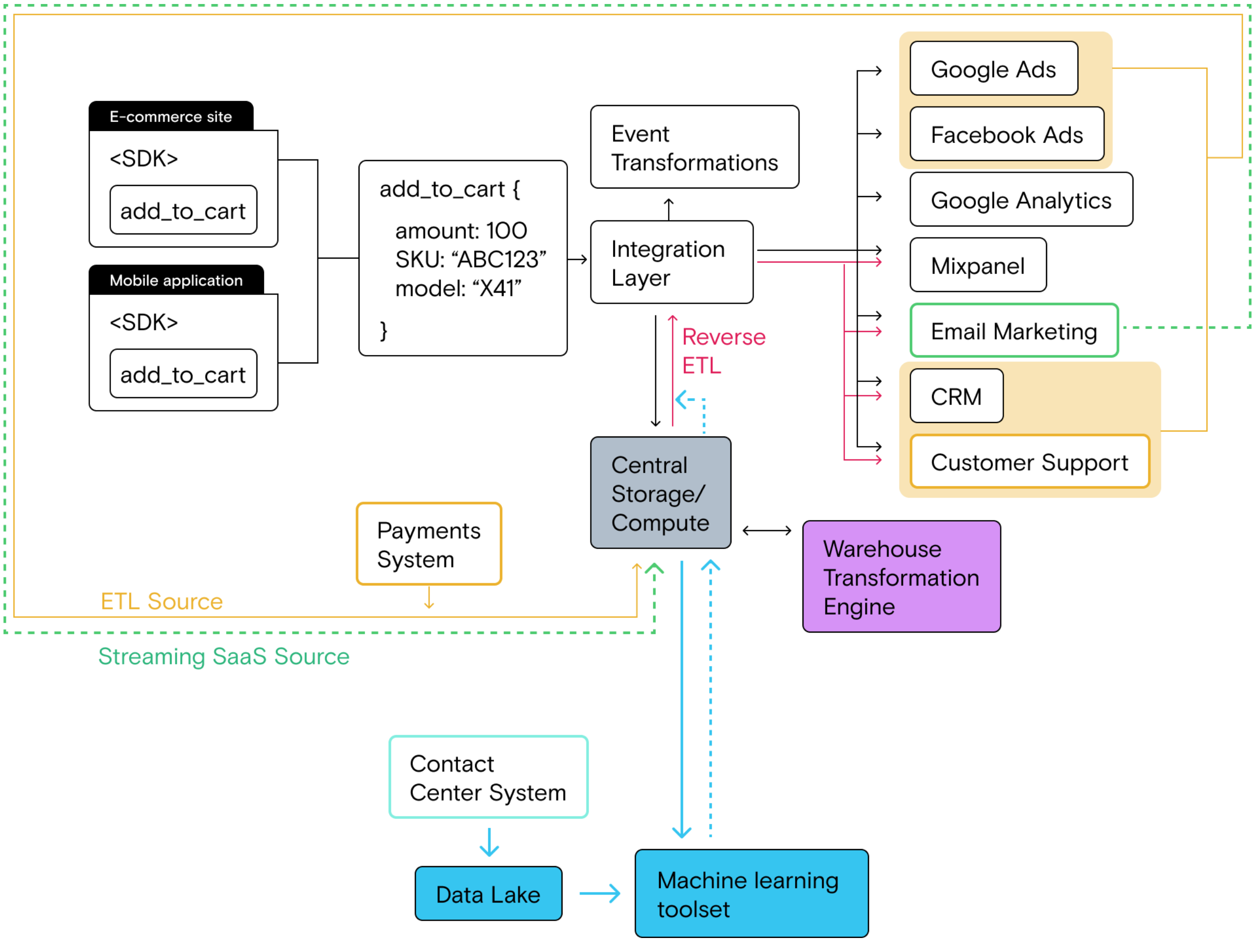
The ML Stack introduces a data flow that enables teams to work on predictive analytics. For companies running the Growth Stack, there are two fundamental challenges:

- **Limitations of warehouse-based analysis:** Data teams hit ceilings in the warehouse when they 1) can't perform their desired analysis in SQL and 2) need to work with unstructured data that can't be stored in the warehouse (more on this below)
- **Operationalizing model outputs:** When models do produce outputs, it can be technically difficult to deliver them to tools where they can be used to optimize the customer journey

The ML Stack solves these problems by:

- Introducing a data lake for unstructured data
- Introducing a modeling/analysis toolset
- Leveraging existing warehouse and reverse ETL infrastructure to deliver model outputs to tools across the stack

As you can see in the architectural diagram below, the ML stack feeds data (inputs) to an analysis and modeling toolset from the data warehouse and data lake. The model runs and produces outputs (or features), which are pushed to the warehouse in the form of a materialized table. That table can then be used in the standard reverse ETL flow to send those features to the rest of the stack.



When is it time to implement the ML Stack?

You'd be surprised at how often machine learning gets thrown at questions that can be answered with basic SQL—something we heard loud and clear from one of the engineers who helped build the data science practice at Airbnb. So, when should you *actually* use machine learning?

The symptoms

The best indication that you need to implement ML tooling into your stack is the desire to make data-driven predictions about the future. Here are a few example symptoms:

- Your teams have built muscle in understanding historical trends and events and why they happened, and now want to act proactively to influence those events before they occur
- This is impossible because teams running the growth stack can't anticipate the likelihood of those events happening, they can only analyze them in retrospect

- You've hit the limits of SQL-on-warehouse analysis and are actively exploring how to apply statistical analysis to your data
- You want to leverage new kinds of data that can't be managed in your cloud data warehouse. This is often due to new tooling or processes that have been implemented (i.e., standing up a call center), or a desire to leverage existing data that hasn't previously been used in analysis

What your company or team might look like

As with every other phase of the data maturity journey, your stack isn't about the size of your company, but your data needs. That said, stepping into the world of machine learning generally requires both a minimum threshold of data as well as dedicated resources to work on predictive projects. This means it's often larger mid-market and enterprise companies that implement the ML Stack. But smaller companies are increasingly collecting huge amounts of data, and the ML tooling itself is getting easier to use (more on this in the tooling section below).

If your company is on the smaller end of the spectrum:

- You likely have a business that produces huge amounts of data (think eCommerce, web3, media, gaming, etc.) but can run with a smaller technical team
- Your business model stands to benefit significantly from predictive analytics, meaning you're willing to invest in machine learning earlier in your lifecycle (this could mean a very high number of transactions for a consumer financial company or significant usage for a mobile game development company)

If your company is on the larger end of the spectrum:

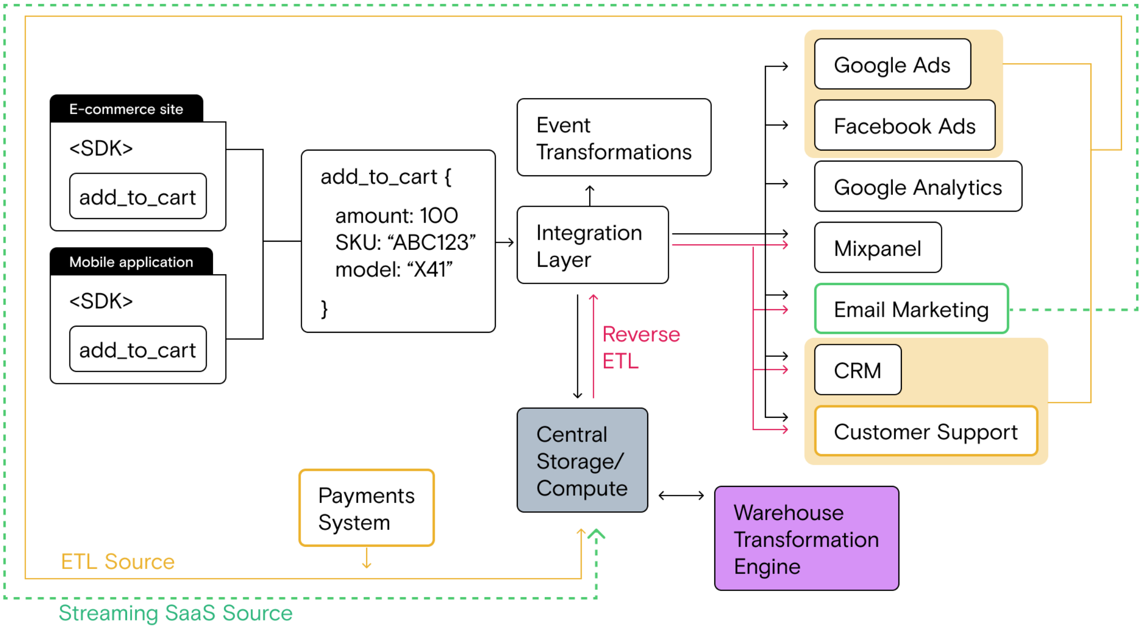
- You've hit the limit of basic optimization across functions and are looking for the next frontier of growth, which requires moving from historical analysis to predictive analysis. This often takes the form of internal initiatives to build out a data science practice
- You've desired to explore data science in the past, but the state of your data and legacy components of your stack have been blocking the effort. But now that you've modernized your stack, you're prepared to step into the world of machine learning
- Your analysts have begun to explore deeper statistical analysis and have already started teaching themselves the basics of machine learning, setting the stage for building formal data science workflows

The return of our example company

For the purposes of example, let’s revisit the fictional company that we’ve been following through their data maturity journey. Here’s a quick recap, with the addition of a new dynamic:

> *You’re an eCommerce company, large or small (as we said above, company size doesn’t matter). Your website, mobile and marketing teams focus on driving digital purchases through your site and app, but you also have a sales team supporting wholesale buyers. Many of the sales team’s prospects are long-time repeat digital purchasers who would benefit from opening an account.*
>
> *After launching a subscription program 6 months ago, your teams have noticed a concerning trend in cancellations. By running multiple analyses on historical data on the warehouse, you’ve discovered that most customers who churn have less than three purchases and have engaged with the customer support team.*

Here’s the current Growth Stack, which includes multiple data sources (event streaming, cloud SaaS streaming and ETL), a cloud data warehouse for centralized storage/compute and multiple pipelines for integrating and activating data (direct integrations fed by real-time event streaming and reverse ETL that sends data from the warehouse to the rest of the stack).

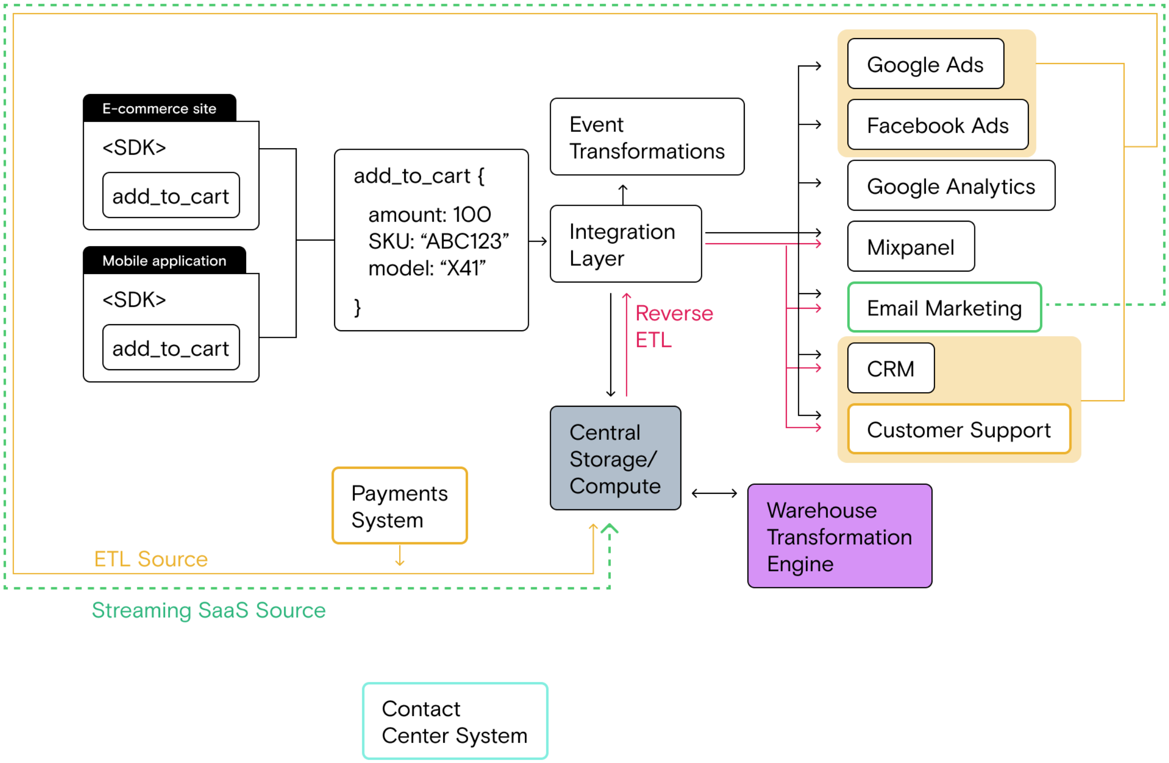


The data stack

At this point, your Growth Stack is a well-oiled machine for data collection, historical analysis and data activation. In fact, this stack is a big part of what enabled your teams to spot the subscription program churn problem.

The latest development in the stack, though, is the implementation of contact center infrastructure to help customer support representatives better serve customers with questions, problems, returns, etc. (This is different from the customer support system, which operates as a ticketing system for asynchronous support requests.)

The contact center system introduces several new types of customer data. First, support agents engage in live chats with customers who initiate contact on the website. Second, support agents have live calls with customers for certain types of support requests. This data lives in the contact center system.



While the contact center data is technically available and could hold the key to solving the churn problem, the Growth Stack isn't suited to working with this data.

Your data challenges

- The contact center data exists as long-form chat transcripts as well as audio files of phone call recordings, both of which are unstructured data that can't be pulled into the warehouse for analysis
    - As a result, even though your historical analysis associates customer support interactions with churn, your analysts can't figure out what it is about those interactions that cause people to cancel their subscription
- Beyond understanding the dynamics of the support interactions, your marketing team would like to know about customers who are likely to canceling so they can send out incentives before the decision is made
    - Your current toolset doesn't support the complex statistical modeling required to make these predictions

The ML Stack playbook: Implementing a simple ML stack

To tackle the challenges above, you will augment your Growth Stack infrastructure with a simple ML toolset to unlock predictive analytics.

Goals of the ML stack

The ultimate goal of the ML Stack is to leverage predictive analytics to further optimize the business by acting before certain events, like churn, occur.

The goals of the ML Stack stage of the data maturity journey are to:

- Introduce a data storage layer that can ingest and manage unstructured data
- Introduce a modeling and analysis toolset that both enables advanced statistical analysis on inputs and produces usable outputs

There is one more piece of the puzzle, which is actually sending the outputs from a model to downstream tools for use—i.e., sending a churn score to the email marketing tool so that offers can be sent to subscribers who are more likely to cancel. The good news is that you already have this delivery mechanism built into the Growth Stack: reverse ETL from warehouse tables. If the modeling infrastructure can push outputs into a table in your cloud data warehouse, you can send them anywhere you want. This is a great example of how the smart tooling decisions you make earlier in your data maturity journey pay off big time as you build more complex use cases.

Data focuses of the ML Stack

ML models require inputs in order to produce outputs. As far as the inputs go, ML models are very data hungry. Thankfully, with the Growth Stack, you have a huge amount of the input data already at your disposal in your cloud data warehouse, including all event stream (behavioral) data as well as relational/transactional data.

The other major piece on the input side is data that the Growth Stack can't manage. So, with one major input covered, the data focuses of the ML Stack are:

- Unstructured data inputs
- Model outputs

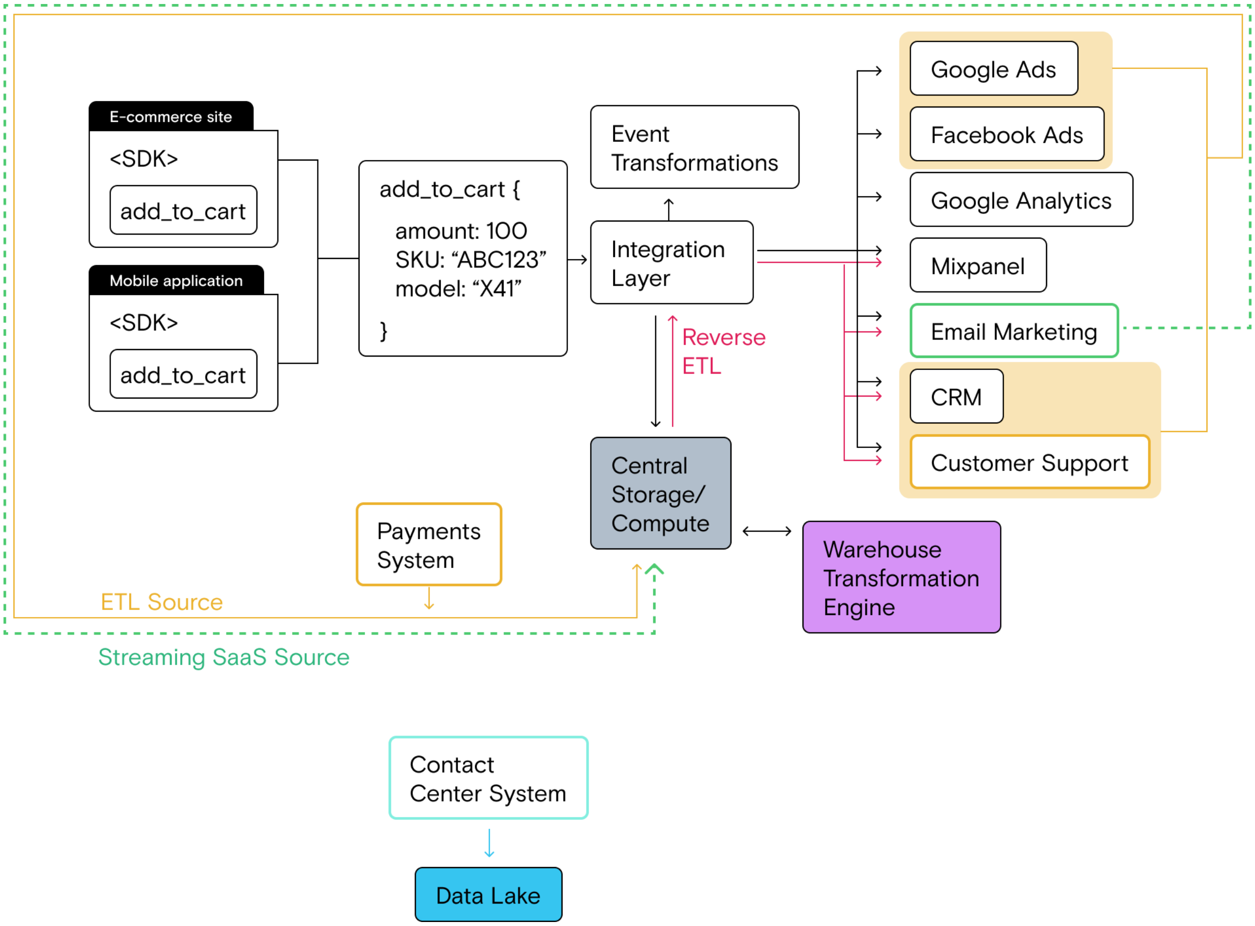
Both are simple on a basic level. Unstructured data inputs (like chat transcripts and audio recordings) can be extremely valuable for building models, especially because you can't use traditional methods of analysis for those data points.

Outputs are exactly what you would expect: The results of the model. These are called *features* in the ML and data science world, and we've already discussed one great example: a churn score.

Step 1: Implement a data lake to collect and manage unstructured data

At a basic level, this is pretty simple. Like cloud data warehouses, data lakes are simple to stand up and most of the major contact center systems have standard integrations with data lakes. If not, there are plenty of third-party tools that can facilitate the loading of unstructured data. The data lake providers themselves have various kinds of loading mechanisms, but these might require a more manual process than an automated pipeline.

Once you've set up a data lake and determined the best method for loading data, your stack will look like this:



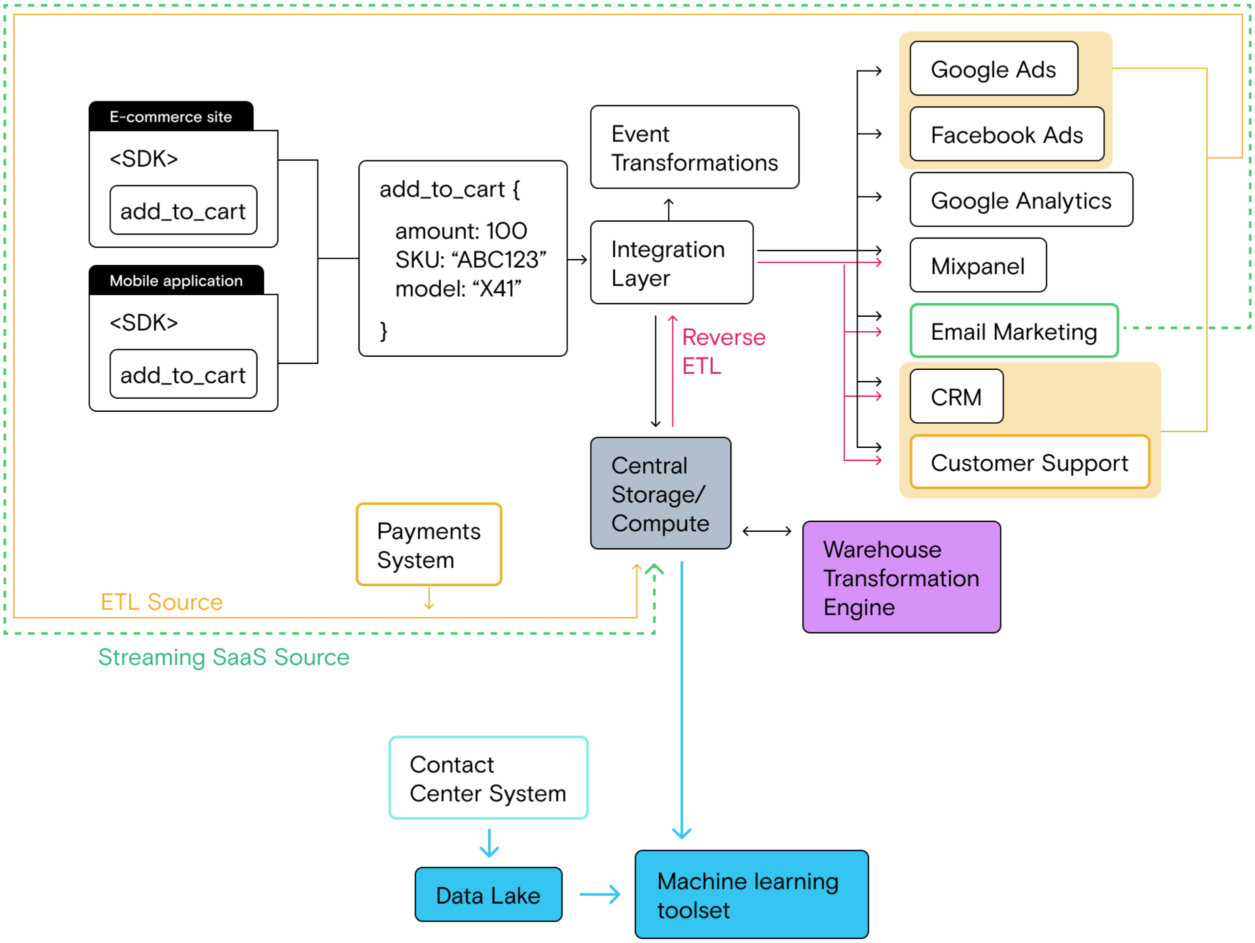
Step 2: Implement a modeling and analysis toolset

As we said above, this isn’t a chapter on ML ops, so we aren’t going to go into a ton of detail on specific tooling. If you’re taking your first steps into ML, your modeling layer could be a simple python/notebook setup that you run locally. On the more complex end of the spectrum, you might choose to adopt a data lake that has ML tooling and integrations vertically integrated—this all depends on your needs and specific tech stack as a company.

Additionally, it’s worth noting that multiple *ML as a service* products have come to market, which allow non-data scientists to run models based on inputs.

No matter which modeling toolset you choose, the fundamentals of this step remain the same: you have to set up the modeling layer and prepare to pull data into it. There’s good news here, too: most modern modeling tools are designed to work with data from multiple sources, so pulling inputs from your data lake and data warehouse should be part of the normal setup process.

Once you set up your modeling toolset and connect to your data lake and data warehouse, your stack will look like this:



Step 3: Develop features and train the model

In machine learning, the inputs you use are specific attributes or input signals that your predictive model uses to generate a prediction. These are also called *features*.

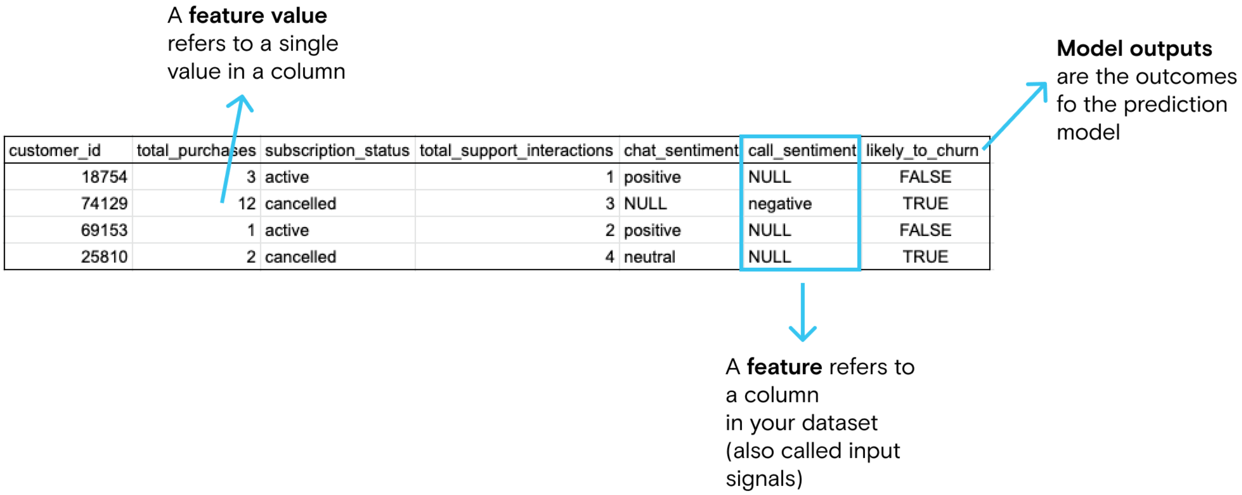
It's worth noting here that selecting features is a non-trivial task.

- If you select the wrong features, your model's accuracy, precision, and recall will suffer due to noisy data
- Selecting features incorrectly may also lead to problems such as data leakage, which can give you great results on paper but perform poorly once deployed

Both are issues your data scientist can mitigate with proper setup and modeling.

Continuing on with our example company, the features for determining churn from the subscription program could be things like:

- Total number of purchases
- Total number of support agent interactions
- Sentiment of most recent support chat or call



A **feature value** refers to a single value in a column

Model outputs are the outcomes fo the prediction model

customer_id	total_purchases	subscription_status	total_support_interactions	chat_sentiment	call_sentiment	likely_to_churn
18754	3	active	1	positive	NULL	FALSE
74129	12	cancelled	3	NULL	negative	TRUE
69153	1	active	2	positive	NULL	FALSE
25810	2	cancelled	4	neutral	NULL	TRUE

A **feature** refers to a column in your dataset (also called input signals)

No matter the specific ML tool you choose, feature development is most commonly accomplished using SQL, Pyspark, or good old Python.

It's worth noting that at scale, *feature stores* become an important component of your ML stack. They make it easy to transform raw data into features, store the features, and serve them both for model training and, once deployed, for delivering predictions to a larger number of destinations, including your own software systems (apps and websites). We will address this in the next chapter on the Real-Time Stack.

Step 4: Train the model

While we won't do a deep dive on ML workflows here, for the sake of completeness, let's cover the basics of training a model.

Once you have defined and computed your features, you need to train your model. The training process requires you to provide the training data (containing the correct output). The training data would include observations, feature values, and the correct answer.

In our example, you want to know if a certain customer will churn or not based on purchase history and the nature of recent interactions with support. You would provide the model with training data that contains customers for which you know the answer (i.e., a label that tells whether the customer churned or not). With this data, your chosen learning algorithm would train the model, enabling it to take features as inputs and predict whether a customer is likely to churn based on those inputs.

Once you've created your model, it's critical to evaluate whether your model is able to make accurate predictions given new inputs (i.e., data not seen during the training process).

If your model performs well, it's time to put the outputs to use, which is where the Growth Stack comes back into the light and saves us a huge amount of work.

Step 5: Send the outputs to your stack for use by downstream teams

Outputs from ML models can be deployed in a number of ways from simple to very complex deployments such as delivering predictions in real-time through your own custom app or website.

Here we're talking about taking your first steps into machine learning, so we'll cover the simplest use case which leverages the existing infrastructure of the growth stack to collect and distribute outputs.

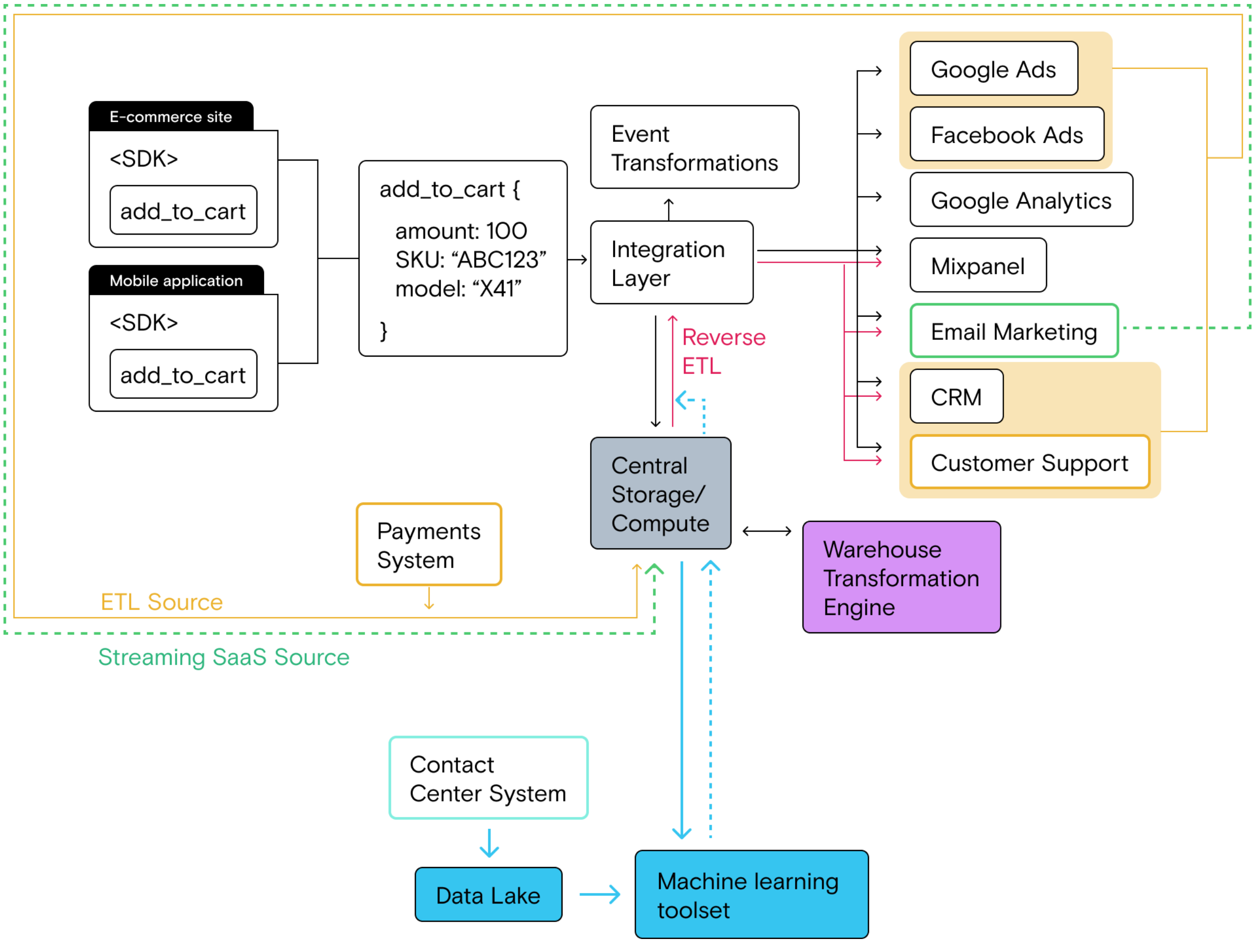
Thankfully, this actually is simple thanks to the cloud data warehouse and reverse ETL pipeline that we already have in place. Here's the basic workflow:

- Push outputs from your ML tooling into a table in your data warehouse
    - Depending on the structure of the output table, you often need to perform a few simple joins to produce a materialized view that has all of the data you want to send to downstream tools
- Send the materialized table to downstream tools via reverse ETL
    - Practically, this means that customers in systems like your email marketing tool will have a new data point added to their contact record (likelihood_to_churn).
    - We generally recommend building important segments in the warehouse as their own materialized views. In the case of our example, this would most likely be a table of users who are likely to churn (likelihood_to_churn = TRUE).
- Celebrate when downstream teams send offers proactively and decrease churn!

It's worth noting here that for the basic use case of sending outputs to your existing business tools, the efficiency of leveraging an existing data flow (warehouse, reverse ETL, downstream tools) saves an immense amount of operational and engineering lift, meaning incredibly quick time to value for your initial machine learning efforts.

Bringing it all together

Once you've got outputs flowing back into your warehouse and through reverse ETL, your basic ML Stack will look something like this:



Tooling and technical review

The ML Stack adds two key components to the stack:

- A data lake (recommended if you have unstructured data)
- Data modeling and analysis tools, e.g., PySpark, NumPy, Pandas, Python
    - These can include things like Python and PySpark, but as sophistication increases, could also include components for model deployment and modeling

The right tools for the job

To take your first steps into ML you won't need to build out a full ML workflow. Here are a few guidelines for selecting the tools you do need:

Data lakes

Like cloud data warehouses, there are multiple good options for data lakes and almost all of the major players are easy to set up and work with. The big decision with data lake infrastructure is whether or not you want a vertically integrated system with built in tooling on top of the data lake to manage some or all of the ML Ops workflow. For companies just starting out, we generally recommend starting with a basic cloud data lake. You'll likely even be able to bring this with you if you migrate to a more vertically integrated ML system in the future.

Data lake... houses?

We'd be remiss not to at least mention the *data lakehouse*. This architecture combines the flexibility and scalability of data lakes with the usability of table-based cloud data warehouses, and it shows a lot of promise for companies who want one tool for both jobs. But lakehouse as a service is still relatively new, and adopting a lakehouse architecture generally requires a larger conversation about your data stack in general. For most companies, the warehouse + data lake setup works great, even at significant scale.

ML tooling

If you're taking your first steps into machine learning, our advice will be the same as it always is: Keep it simple, even if that's basic Python and notebooks. Your data scientist is probably already comfortable with certain languages and tooling—start with those.

As your data science practice matures, the team can make bigger decisions about more complex and comprehensive ML workflow architecture. The thing to keep in mind here is that if you've chosen an extensible toolset at every phase of modernizing your stack, you should be able to easily test and change out ML tooling around your warehouse and data lake as your team builds preferences and you determine what will work best for your specific needs.

One additional thing to bring back up is the rise of *ML as a service* tooling that provides all of the modeling and training as a service. There are even services that can run directly on your warehouse, meaning feature development can happen in your native SQL editor. Depending on the structure of your team and your machine learning needs, this can be a low barrier way to enter the world of predictive analytics.

Outcomes of implementing the ML Stack

The ML Stack is the natural evolution of the Growth Stack. While the Growth Stack allows you to derive insights using descriptive analytics, the ML Stack unlocks entirely new frontiers with predictive analytics. Implemented correctly, the ML Stack can drive massive bottom line impact by decreasing churn and increasing lifetime value.

Specifically, teams can:

- Mitigate problems proactively, as opposed to reacting after they have been observed
- Leverage valuable unstructured data that wasn't possible to use before

With your built-in warehouse + reverse ETL distribution system from the Growth Stack, you'll have far quicker time-to-value from your models than if you had to build all of the deployment componentry yourself (which is always great to tell your boss 🙂).

Signs that you're outgrowing the ML Stack

For many companies, the ML Stack is the ideal end-state solution. However, for B2C companies with millions of customers and commensurate marketing spend, there is an opportunity to drive even more value from machine learning outputs through real-time personalization. If you can modify the customer experience in real-time based on outputs from a machine learning model, you can optimize the customer journey on a micro-level and remove friction at every possible point.

If you're at a stage in your data maturity where you're beginning to explore real-time personalization, you will need additional tooling to upgrade your infrastructure to the Real-Time Stack.

Chapter Four

The Final Frontier: Real-Time Stack

Before we go on to explore strange new tools. To seek out new use-cases and new optimizations. To boldly go where no man or woman has gone before. Let's do a quick recap of how far we've come in the data maturity journey. In the Starter Stack, you solved the point-to-point integration problem by implementing a unified, event-based integration layer. In the Growth Stack, your data needs became a little more sophisticated—to enable downstream teams (and management) to answer harder questions and act on all of your customer data, you needed to centralize both clickstream data and relational data to build a full picture of the customer and their journey. You solved these challenges by implementing a cloud data warehouse as the single source of truth for all customer data, then using reverse ETL pipelines to activate that data.

As the business grew, optimization required moving from historical analytics to predictive analytics, including the need to incorporate unstructured data into the analysis. To accomplish that, you implemented the ML Stack, which included a data lake (for unstructured data) and a basic machine learning tool set that could generate predictive outputs like churn scores. Finally, you put those outputs to use by sending them through your warehouse and reverse ETL pipelines, making them available as data points in downstream systems (i.e., email marketing, CRM, etc.).

Setting the stage

For many companies, the ability to act proactively based on future predictions is game-changing, and it's the last stop on their data maturity journey.

But for some companies there are use cases where pushing attributes to downstream tools in batches isn't enough, in large part because the use cases are often limited to ad-hoc email or text message blasts (or, on the advanced end of things, more intelligent marketing automation).

The holy grail of the machine learning stack for those companies is personalizing interactions with every customer and, where possible, customizing their experience in real-time. At RudderStack we often call real-time personalization the “final frontier” of using customer data.

This chapter will be a bit different than the last three because, as crazy as this sounds, very few companies have actually built an effective, end-to-end real-time personalization engine. Most of the popular success stories come from companies with thousand-plus person engineering teams and unlimited resources to build custom tooling.

That technology has begun to trickle down, though, and today it’s possible for many more companies to build powerful real-time personalization use cases by using the right tools.

Still, there’s a high level of complexity on the software engineering side in order to modify user experiences in apps and websites in real time—what we call the “last mile” of delivery. Delivering the last mile also varies significantly for every company due to their proprietary code and software infrastructure.

To that end, this chapter will be less prescriptive and instead explore suggested architectures for use cases where the output of ML systems can be leveraged in real time. Our goal is to help data engineers think creatively about how they could enable real-time use cases in partnership with other teams.

Defining real-time

The ML Stack we outlined in the previous chapter allowed us to provide inputs to a model and use the outputs. Because timeliness wasn’t a primary concern, a batch job on some schedule was sufficient. The goal was to have the ability to take action proactively based on predictive user traits (i.e., likely_to_churn). Also, the destinations for the outputs were cloud SaaS tools used by downstream teams like sales, marketing and customer success, which made sending updated user traits easy (because we already had the integration infrastructure set up).

The Real-Time Stack delivers on the same basic outcome of modifying the user journey but takes it two steps further:

- First, this stack enables use cases where delivering personalized, real-time experiences is critical
- Second, this stack delivers results directly to the user experience in the website or app, literally modifying and customizing what they see

What do we mean by real-time?

Much like the term machine learning, real-time is an often mis-understood and abused term in the world of data. For some companies, going from multi-day jobs to 1-hour jobs for key reporting can feel like a massive jump to “real-time” data. For other companies, real-time means real-time in the literal sense, where something is modified on a sub-second timeline (often measured in milliseconds).

In this chapter, we will consider use cases where the user experiences personalization in real-time, but, as you might expect, that experience can happen in different ways depending on the user's actions.

Two flavors of real-time personalization

In general, real time personalization comes in two flavors:

Next behavior personalization

This use case delivers personalized results when the user takes some future action. TV streaming services are a great example: You don’t know exactly when a user will log in next, but when they do, you want to serve them the most relevant content possible based on what they’ve recently consumed.

This means that the user experience is modified in real time from ML outputs that are accessed in real time. However, the entire data loop doesn’t have to run in real time because there is some amount of time between user actions. Said another way, this is real-time personalization for the user, but not necessarily real-time ML.

In-session (or next-page) personalization

This use case is the bleeding edge of real-time personalization. It involves collecting inputs from a user in real time, routing them through an ML model that runs in real time, and using the model’s outputs to customize that particular user’s experience while it is happening. This requires the data loop itself to run in real time (or, practically, on a sub-second/millisecond timeline). One good example of this is real-time personalized search results. When a user searches your site or app using certain terms, you want to provide recommended results that they are more likely to purchase. Those ML-derived recommendations need to be calculated and delivered when the search is executed before the results load.

Below we'll cover examples of each use case, but first, let's look at how we'll augment the ML Stack to make the Real-Time Stack.

Real-Time Stack overview

The Real-Time Stack introduces a few new components that enable real-time delivery of outputs back into apps and websites.

Remember, in this phase data tooling is one part of the equation. The last mile of delivering a personalized experience actually occurs in your software–be that a website or app–so achieving these use cases requires collaboration across data engineering, data science, and product engineering teams.

One significant implication of this architecture is that your product engineers have to write custom code in your software that accesses the outputs from your ML models to incorporate them into the user experience. Building this last mile has gotten easier, but it varies depending on the company's stack. This is why is why we are focusing on the data flows in this chapter.

The Real-Time Stack addresses two fundamental challenges to delivering real-time experiences:

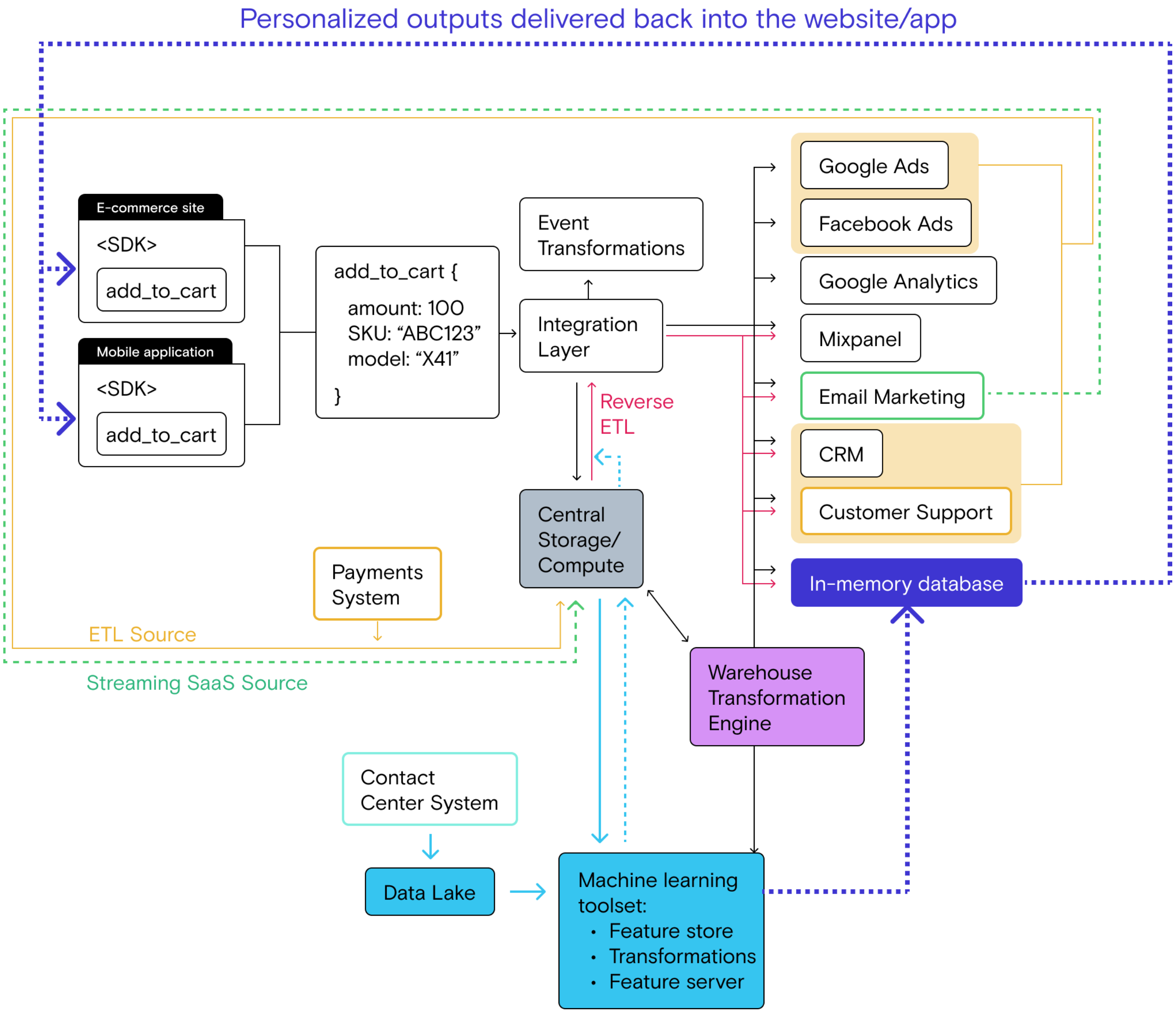
- **Data freshness:** Real-time personalization requires inputs that are happening right now, meaning you can't wait for compute jobs to run or tables to be materialized, then delivered
- **Latency:** How do you activate the output of your ML model while your customer is using your app?

The Real-Time Stack solves these problems by:

- Introducing an online, in-memory store that makes outputs available in real-time
- Introducing more robust machine learning infrastructure to enable modeling with real-time inputs

As you can see in the high-level architecture below, the Real-Time Stack looks similar to the ML stack, with the addition of the in-memory data store. What's not represented here is the work that goes into the dashed purple line connecting ML outputs with the app and website themselves–the last mile delivery.

Also note the additional machine learning toolset: A feature store, transformation functionality, and a feature server that can serve outputs in real time. The ML box in the diagram intentionally over-simplifies the ML setup becase we aren't digging into the specifics of ML ops systems here.



When is it time to implement the Real-Time Stack?

We’ll be clear at the outset: Some companies will never need to build the infrastructure required to deliver experiences in real time. So before you get started, remember that avoiding the temptation to over-architect your stack always pays dividends. That said, for some companies the Real-Time Stack is imperative for driving additional growth. Let’s take a look at practical indicators that you might need to implement this infrastructure.

The symptoms

Similar to the ML Stack, the need to optimize key points in the customer journey is the trigger for exploring real-time use cases. That said, implementing architectures like this one make sense only when you've exhausted the utility of other kinds of personalization efforts. Here are a few example symptoms:

- Relatively small percentage gains (i.e. >1%) in key metrics like retention or churn mean significant impact on the bottom line of the business
- Downstream teams have hit the ceiling of personalization features available via batch delivery of outputs activated through their SaaS tools–conversion rates are as good as they will get without introducing the element of timeliness and modifying the app/website experience
- You work in a business that is particularly sensitive to timeliness such as delivery, media, eCommerce, or IOT

What your company or team might look like

Up to this point in the Data Maturity Journey, we were careful to make the point that each stack might be used by a company of any size. However, the Real-Time Stack generally only makes sense for larger companies because of the dedicated product engineering investment required to deliver the last mile. Most often that means consumer brands with millions of monthly customers or users.

If you're exploring the real-time stack, you likely:

- Have a mature data science practice operating in your company
- Have access to significant engineering resources
- Have implemented successful A/B testing programs at scale in your website or app

Returning to our example company

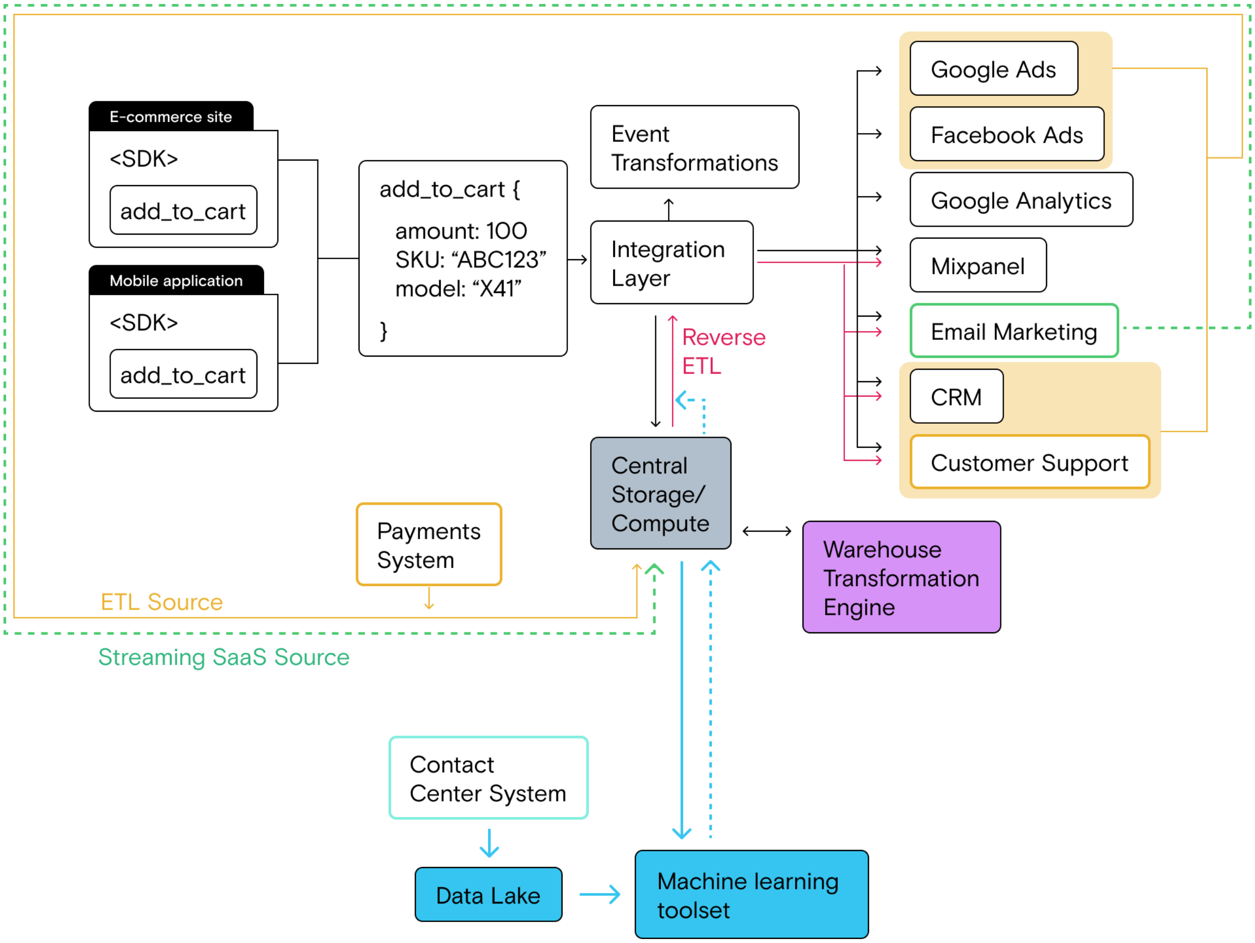
Let's return to the example company we've been following to see how they're approaching the need for a Real-Time Stack:

> *You're now a large eCommerce company.*
>
> *Your website, mobile and marketing teams focus on driving digital purchases through your site and app, but you also have a sales team supporting wholesale buyers. Many of the sales team's prospects are long-time repeat digital purchasers who would benefit from opening an account.*

After launching a subscription program 6 months ago, your teams noticed a concerning trend in cancellations. By leveraging the ML Stack, you were able to flag customers who were likely to cancel, enabling the marketing team to reduce churn by emailing offers proactively.

Excited by the success of using predictive analytics to reduce churn, the marketing team now wants to explore delivering real-time recommendations to various users as they interact with the site and app. A sharp data analyst does an analysis showing that a 1% increase in basket size could drive $10M in revenue - a much needed boost in a highly competitive market.

Here's the ML Stack you're currently running, which introduced a storage layer for unstructured data and a modeling and analysis toolset, allowing the company to unlock predictive analytics:



This ML Stack is great for delivering model outputs in batches that aren't time-sensitive, but it can't facilitate real-time use cases.

The data stack and the data challenges

In the previous phases of the data maturity journey, the introduction of new systems and data types created challenges that required you to implement additional tooling. In the case of the Real-Time Stack, there aren't new systems producing net-new customer event or relational data.

The primary concern of the Real-Time Stack is the computation and delivery of ML outputs directly into your app or website in a timely fashion. At this point you aren't mitigating pain, you're seeking to capitalize on untapped opportunity.

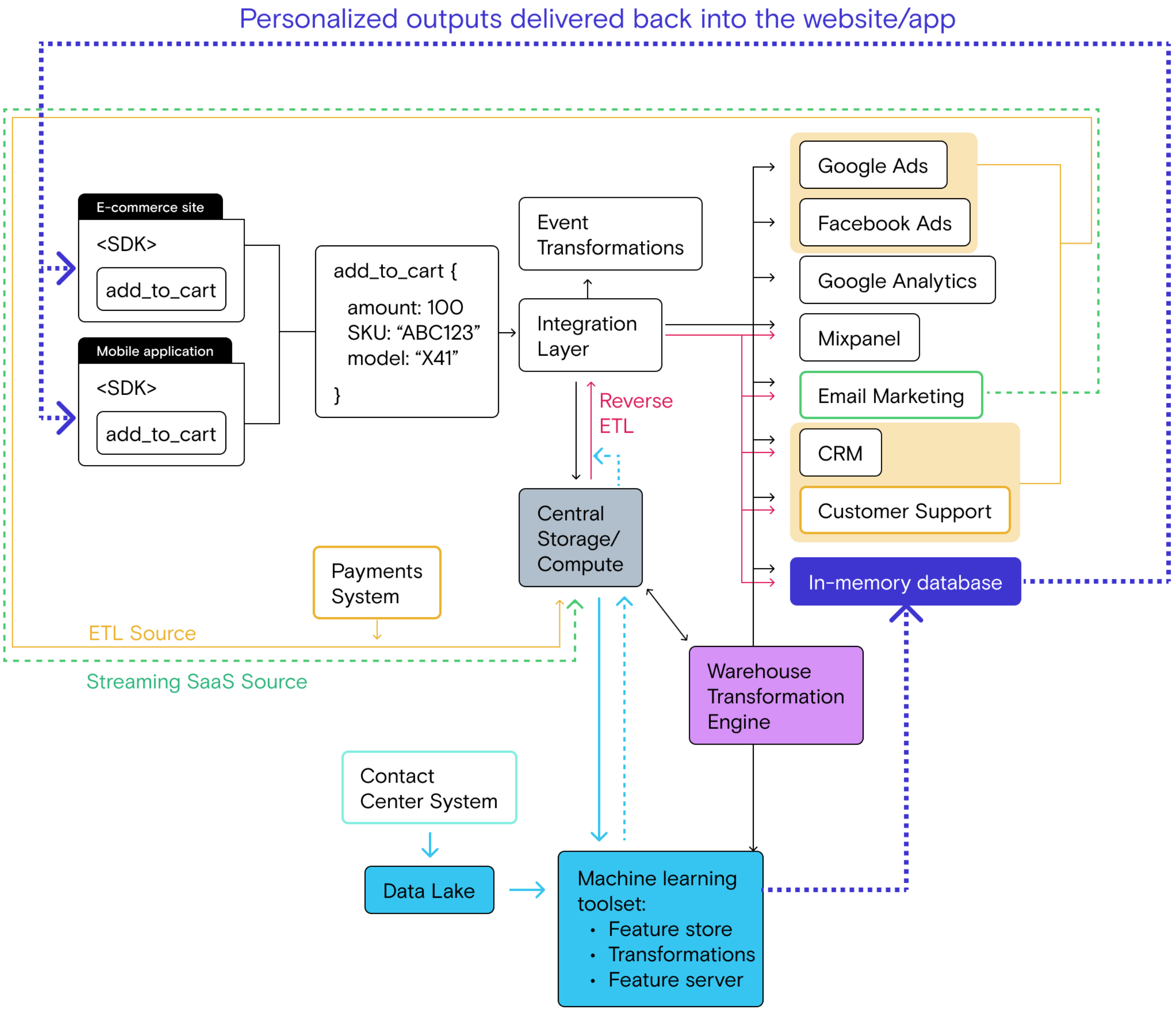
In order to solve the delivery problem, there are several new components that need to be added to the stack:

- **A real-time, in-memory data store:** This a repository of outputs made available for consumption in real-time (most often via an API on top of the in-memory store)
- **More robust machine learning tooling:** In order to enable real time, our ML toolset needs to include:
    - A feature store (for easy access to inputs)
    - A transformations layer to make developing and working with features easier
    - A feature server, which can run analyses in real time and deliver outputs to the in-memory data store
- **Last mile code that can consume outputs in real time:** This is code in your website or app that can access the in-memory data store to pull outputs and incorporate them into the customer experience

In practice, the transformations layer and feature server often operate as a single *model serving layer* that takes care of both functions. We've opted to break the functions out in this chapter so readers don't get caught up in nomenclature.

Again, we're intentionally leaving out a significant amount of detail here. You can implement all kinds of ML tooling and last-mile code depending on your specific stack and needs. Our goal is to highlight the high-level data flows as they relate to the existing infrastructure of the Growth Stack and ML Stack.

Here's an example of what this architecture could look like:



One thing to note about this proposed architecture is that the in-memory database is fed by multiple sources: The real-time event stream, reverse ETL pipelines, and the ML toolset. In a sense, the in-memory store takes your full view of the customer and makes it accessible to other parts of your stack in real-time.

The Real-Time Stack playbook: Two example data flows

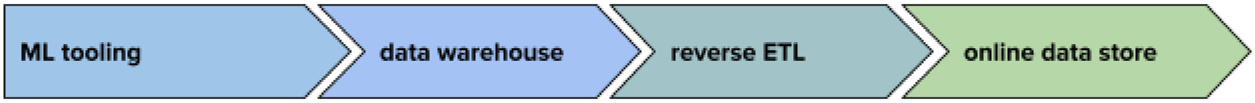
Instead of diving into the specifics of implementation, let's dig into the two primary data flows mentioned above to explain the different data flows required to deliver each.

Use-case 1: Recommendation engine (batch computation, real-time serving)

This use case enables next behavior personalization. Going back to our example company, you want to personalize their experience on the next visit or log-in to increase the value of a user's potential transaction. Specifically, you want to drive additional purchases by recommending products at the top of the page (ML outputs) that your customer is likely to purchase based on what you know about them, such as purchase history, purchase frequency and demographics (ML features).

In this case, the model features can be computed from historical data (vs. real-time data) and reflect likely customer preferences. The primary goal is making the outputs available to serve in real time when the user takes their next action.

Because there is some amount of time between user actions, you have the luxury of leveraging much of your existing infrastructure to deliver an experience that is personalized in real-time for the user. Practically, this means that you're computing outputs in batch on some schedule and pushing them to the online store to be consumed. If this sounds familiar, it is. This is the exact same flow from the ML Stack:

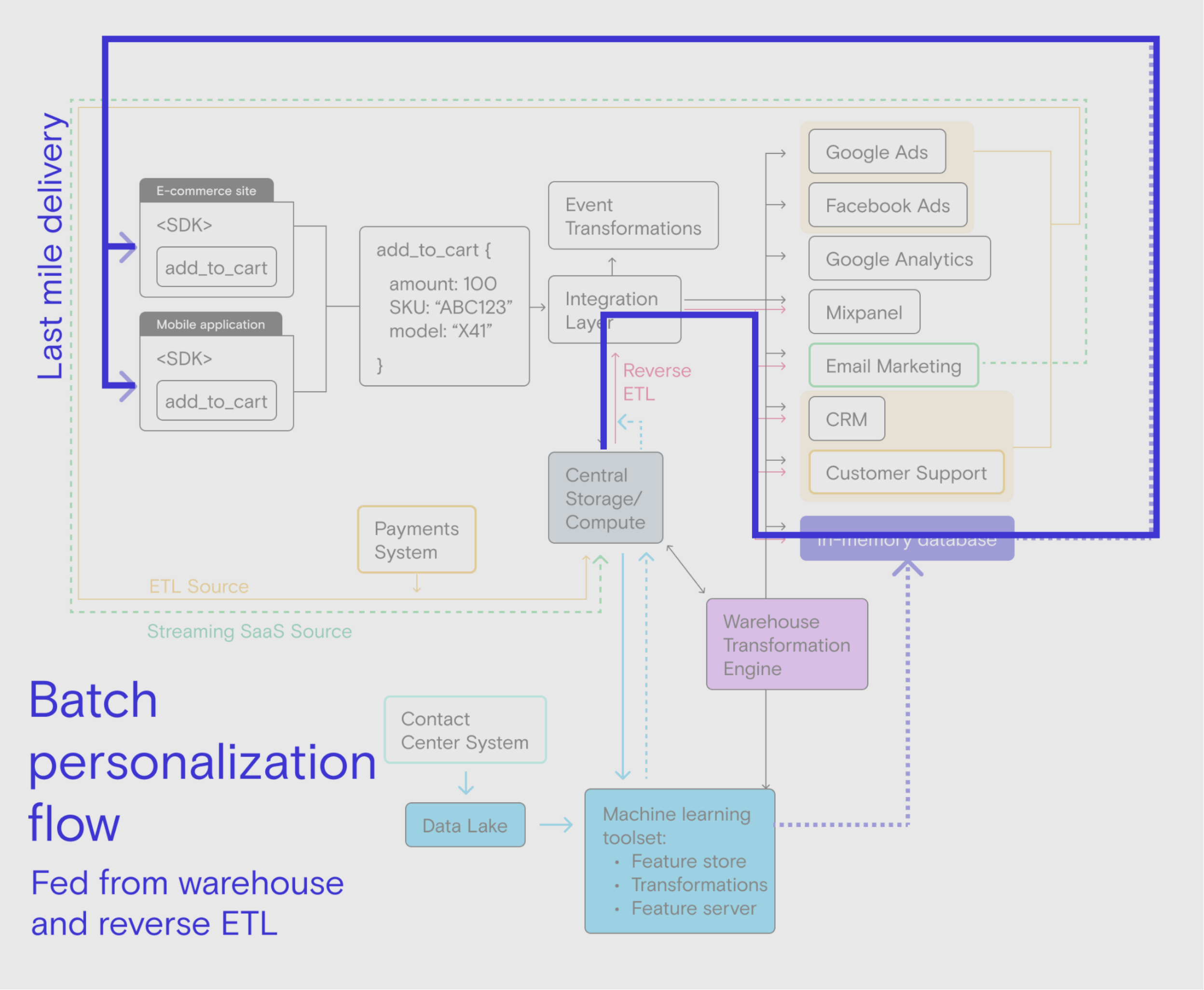


The primary difference is that the ML outputs are made available in some kind of an online store that can be accessed when the user next visits or logs in (you experience this data flow as a user every time you log into services like Netflix). The online store is required because data warehouses and data lakes, where your batch outputs are stored, aren't designed for real-time online lookup. Also, depending on their data types and stack architecture, some companies choose to route outputs through their data lake instead of their warehouse.

As we said above, this use case enables real-time personalization, but doesn't require real-time ML. Just because it's a batch flow doesn't mean that it can't run fast, though. We've seen customers run certain batch jobs like this on a 15-minute schedule for their personalization use cases.

The other thing to note about next behavior personalization is that it doesn't require additional ML tooling. You can use the same modeling mechanisms, you just deliver the results to an online store in addition to the data warehouse. Next behavior personalization is also easier to implement in your app or website, another reason many companies start here for modifying in-app or web experiences.

Here's what this high-level data flow looks like:



Use-case 2: In-session personalization (real-time computation, real-time serving)

There are certain use cases where even a fast cadence of batch jobs isn't adequate, and real-time ML computation is required. Let's look at a few examples.

Returning to our example company, let's say we want to deliver recommendations in search results based on the terms typed by the user. In order to deliver personalized recommendations in the results, we would need to:

- Feed the search terms as inputs into the ML model
- Run the model and produce outputs
- Incorporate the outputs into the search results in the website or app

…all of this between the time the user clicks “submit” and the search results load. This is real-time personalization and real-time ML.

Another great example is fraud detection. Let’s say you are an online bank and you want to detect fraudulent activity based on the end user’s browser properties and browser behavior. This requires the ML system to make predictions based on the current live user activity and user traits and run a model on those inputs in real time.

Real-time computation may also be required where it is impossible to make batch predictions for all possible combinations of inputs. Think about a use case for showing a user similar products to the one they are viewing. Let’s say we want to tune the similar product recommendation engine not just based on the last product viewed/clicked by the user in-session, but also the user’s *historical* behavior. If we were to use a batch-ML system for this scenario, it would require pre-computing the output recommendations for every combination of product and user options. With a few thousand products and 10 million users, this would require tens of billions of combinations. Pre-computing predictions for that many combinations is not only infeasible, but storing the computed results would be prohibitively expensive.

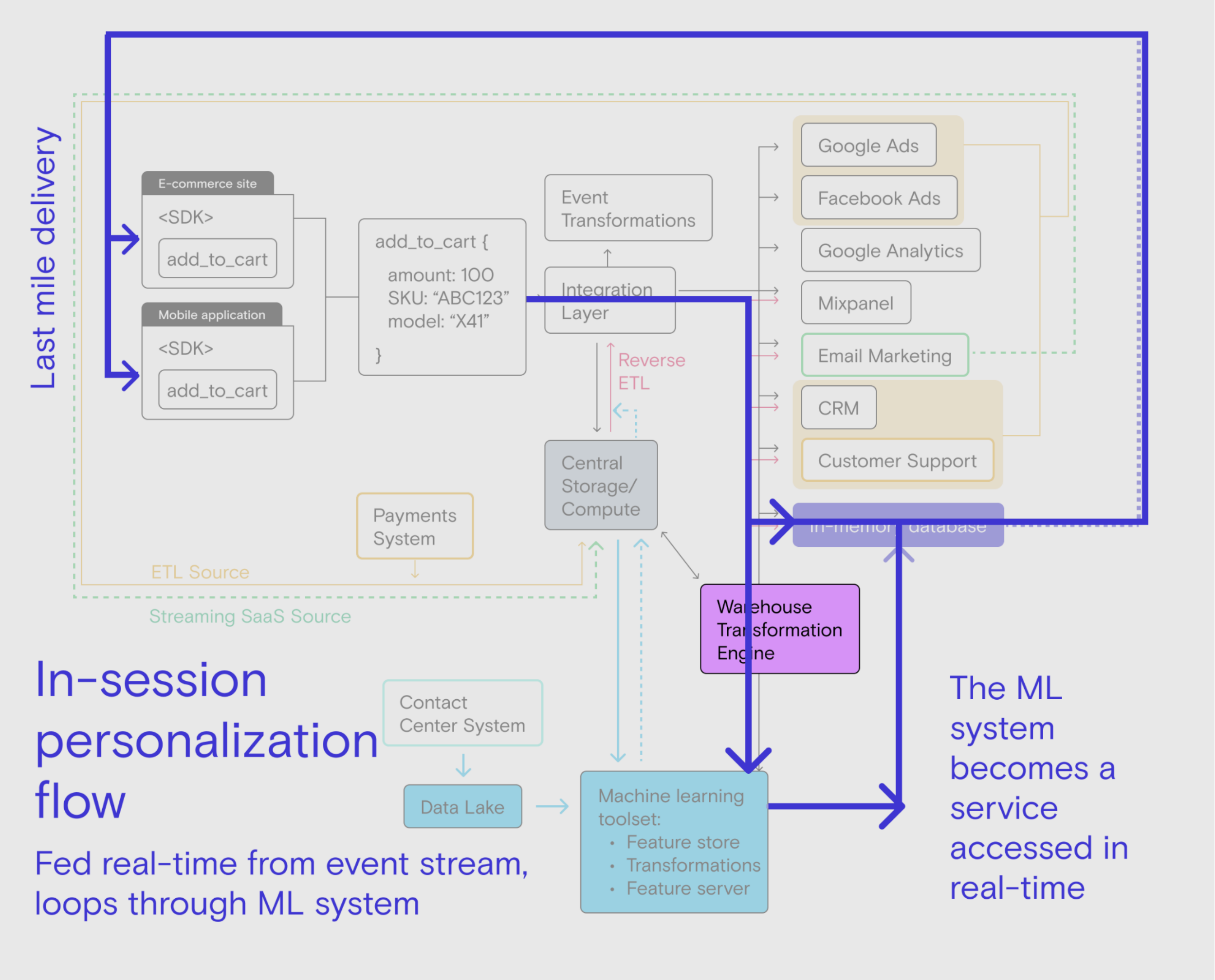
A much better approach is to design an ML system takes the user’s historical profile and their live behavior as input in real-time and returns relevant products for that specific combination.

One small step for personalization, one giant step in infrastructure

While the jump from batch ML to real-time ML may feel seamless to the end user, the system required to achieve in-session personalization is substantially more complex.

First, let’s look at the high-level data flow that can feed an ML system with live data. Then we’ll dig a little deeper into the additional ML system functionality required for real-time computation.

Here’s the data flow for real-time ML:



The most important thing to note about this architecture is that the ML toolset itself must become a service that can be accessed in real-time. This makes it possible for the ML system to receive real-time data and make the real-time computations required for delivering in-session personalization.

To explain this further, let's take one step deeper into the machine learning tool set box in the diagram in the context of our "show similar product" recommendation engine.

Our example requires two data sets as inputs:

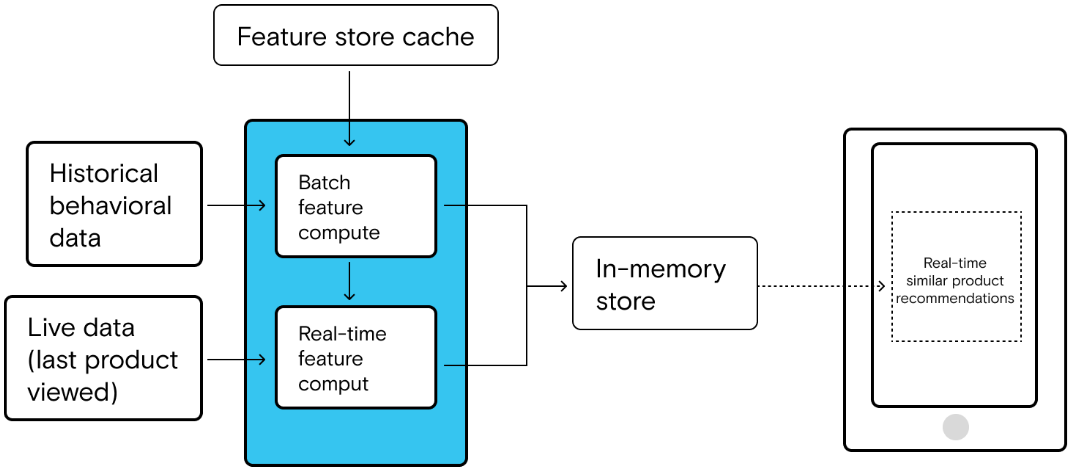
- Historical user behavior
- Live data on the products or links the user just viewed/clicked

The basic data flow for the live data piece is straightforward once the ML system is set up as a service. Real-time behavioral data is fed to the ML system from a real-time event stream. If you've built your stack following the data maturity journey, you already have this in place.

The historical data is trickier because that data has to be collected and updated on some interval, meaning that it runs as a batch job on some schedule (for example, historical behavior up through the last 6 hours). But the data also needs to be available for use by the ML system for the real-time computation.

To make the data available, the batches must be sent to an online cache that is readily accessible by the ML computation service. In the ML workflow, this is often called a feature store.

In the full flow, a user's real-time pageviews and clicks are fed into the ML service in real time. The ML service then combines these inputs with inputs from the feature store cache in order to run the full model and deliver the outputs—all in real time. Here's a high-level overview of what this specific piece of the architecture might look like:


Feature store cache
Historical behavioral data
Live data (last product viewed)
Batch feature compute
Real-time feature comput
In-memory store
Real-time similar product recommendations

Tooling and technical review

The Real-Time Stack adds three key components to the stack. One is a key piece of the puzzle no matter your stack, while the other two are highly subjective based on your specific tool set and product software infrastructure:

- An Online, in-memory data store (required to make outputs available in real time)
- An ML toolset enabling ML computation as an online service
- Software infrastructure in your app or website to access outputs in the online data store

Outcomes from implementing the Real-Time Stack

While it's technically complex to implement, both from an ML and last-mile delivery standpoint, the Real-Time Stack is delivers the infrastructure to modify any part of the digital journey in your websites and apps in real-time. Our examples focused on things like increasing basket size, but the use cases are endless. They span the spectrum from fraud detection to feature adoption and beyond, and when these projects suceed, they can significantly impact the bottom line.

The data maturity journey is never over

Thanks for taking this journey with us! We hope the content has helped you cut through industry buzzwords to better understand the practical fundamentals of data stack architecture.

Remember, the data maturity journey is continual. New use cases will arise, and new tools will be adopted by downstream teams. Your data tooling itself will change, new flows will be unlocked, and occasionally tools will become outdated. Through it all, your goal as a data engineer holds steady–to build and run the most flexible, efficient stack to meet the needs of your business today while preparing for the needs of tomorrow. To infinity and beyond 🚀

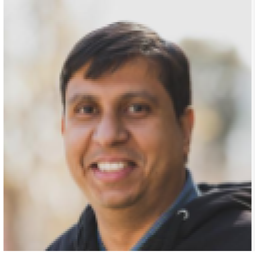
Author

Eric Dodds

Head of Product Marketing at RudderStack

www.rudderstack.com

Contributors



Soumyadeb Mitra
Founder & CEO at RudderStack

Dileep Patchigolla
Principal Data Scientist at RudderStack

Ben Rogojan
Seattle Data Guy, Data Science & Data Engineering Consultant
theseattledataguy.com

Max Werner
Owner at Obsessive Analytics, Data Analytics and Infrastructure Consultant
obsessiveanalytics.com