APIFUSE

The Definitive Guide to Building Product Integrations

- Build vs. Buy

Once you recognize the need to offer integrations to your customers but don't want to rely on third-party services like Zapier, then you must decide whether to build or buy an integration solution. This article explores the total cost and considerations of both approaches.

You must decide whether to build or buy an integration solution.

There is no single cookie-cutter integration that will suit the needs of every one of your SaaS applications' end-users. Each end-user will want custom functionality. This guide will help inform your business and R&D team as to the time it will take to build an integration solution in-house and how to best scale your integration strategy based on the collective knowledge of a team of experts with years of experience in this domain and use case.

8 Crucial Components of a Flexible Integration

Eight main components need to be developed in order to achieve a flexible integration offering for your customers. This article explains each of these components in detail to guide your decision-making process.

Simple	Moderate	Complex
Connectors	Connectors	Connectors
Authentication	Authentication	Authentication
Schedulers	Schedulers	Schedulers
Webhooks	Webhooks	Webhooks
Monitoring & Alerting	Monitoring & Alerting	Monitoring & Alerting
Testing & Debugging	Testing & Debugging	Testing & Debugging
—	Orchestration	Orchestration
—	—	Mapping & Transformation

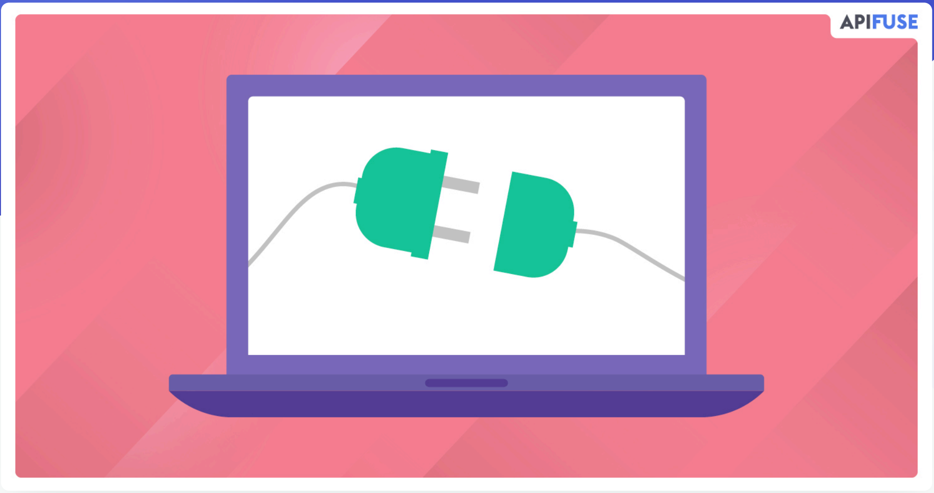
Before diving deep into each component, let's understand the following levels of complexity:

Simple: The integration has a single trigger and single action with no custom logic involved.
Moderate: The integration has a single trigger and can kick off a series of different actions.
Complex: The integration has a trigger, kicks off a series of different events based on custom logic and can transform the output at each step of the workflow (Date/time, branching logic, if/then, data types and formula fields.

Now, let's break each of these components down further:

Connectors

Every SaaS application category (i.e CRM) has a plethora of web services (applications) available that your customers will want to integrate with. For example, if your product needs to provide integrations for accounting software, then you have to consider QuickBooks Online/Desktop, FreshBooks, NetSuite, SAP Concur, Sage, and the list goes on. If you want to capture a large number of systems that your customers use then you must build connectors for the maximum number of applications in each business category relevant to your customers.



Building connectors for all these applications is not an easy task. Each application uses different standards and methods when exposing their services. For example, QuickBooks Online exposes data using a REST-based API secured with OAuth 2.0, SAP exposes data using an OData API secured with Basic authentication, and NetSuite exposes data using a SQL operation and authenticated using a JDBC connection string.

Some applications also require you to be in their partner program to use their API. For some applications, you even need to purchase the license to be able to use their API. This can be a time-consuming effort on top of the actual development effort.

Main Considerations:

1. HTTP standards and patterns

- There are many different HTTP standards now in use like REST, SOAP, OData, and GraphQL.
- Each pattern dictates how you retrieve data from third-party applications.

2. Request/Response formats and Schemas

- You must understand the different request/response formats used by the third-party application. Most of the new applications use JSON payloads but legacy applications use XML/SOAP payloads.
- You should be able to handle these various data formats and mediate between them by converting these different data formats to a single data format.
- Third-party APIs are susceptible to change. API versions may change or old APIs may be deprecated. API versions request/response schemas may change, which could break your existing integration and impact your customers.

3. Pagination

- When you receive large data sets from third-party applications, you are likely to use pagination.
- Pagination is mostly used in polling-based triggers and sync operations where you would query the APIs based on a specified date range.

4. Throttling and API Limits

- Many third-party applications put limits on API consumption. Those restrictions can include per account, per partner, and per resource. For Example, Google Sheet has an API limit of 100 requests per 100 seconds per user and 500 requests per 100 seconds per project (The project being you).
- When you build connectors you have to throttle API requests as per the third-party API limits which again, can be a challenging task and additional development work.

5. Common data model

- When you build connectors for similar apps you might want to build a common data model. For example, Google Drive, One Drive, Box, and Dropbox are similar apps being used for the same purpose. You could build the common data model (also now known as unified API) for these so that you can simplify integration data mapping.

Estimated Time:

4-8 weeks per connector

Authentication:

As explained in the connectors section, each application uses different authentication mechanisms. You have to carefully design your back-end and front end for supporting different types of authentications.

Main Considerations:

- There are 10+ authentication mechanisms available and each one requires its own implementation.
- At the minimum, you should support the most widely used authentication mechanisms which are Oauth 2.0, Oauth 1.0, OpenID Connect, API key, Basic Authentication, and HMAC signature.
- Oauth 2.0 & Open ID connect itself are separate applications because you need to build front end components for initiating the authentication and redirecting the user post-authentication. You also need a backend service to receive the authentication codes and exchange them for access/refresh tokens. You also need to store those tokens securely.

- HMAC signature-based authentications are mostly for custom authentication types. There are no standards for these response types. For example, some applications may expect you to hash the entire payload including the header/request URLs with shared keys while others might expect private keys. Some expect you to include timestamps while others do not. The point being, most of the time you will need to do this authentication for every new application that uses HMAC.

Estimated Time:

1-2 months
(this is a separate microservice you must develop).

Schedulers:

Schedulers or polling services are required for pulling data from third-party applications on a date/time range. The purpose of a scheduler is to invoke a third-party API periodically (cron job) to query datasets changed in a particular period and if any data is found it will trigger subsequent steps (actions).

Main Considerations:

- Building a cron job for a single user may seem trivial, but when it comes to 100s or 1000s of integrations, your pollers should spawn 1000s of cron jobs.
- You should be able to spawn cron jobs when the integration is published and removed when an integration is deleted.
- To support these requirements, you need to maintain the state of the cron job by using an in-memory cache or database while exposing microservices for spawning and killing cron jobs.

Estimated Time:

1+ month

Webhook:

A webhook is an event or data pushed from a third-party application in which you need to expose a URL and configure it in a third-party application so that the third-party application can send the event to that endpoint whenever data has changed. Building a webhook again might seem trivial but supporting webhooks for multiple applications will become challenging.

Main Considerations:

- There is no fixed standard for how a webhook can be used.
- Some applications encrypt the request payload while others don't.
- Webhook authentication/verification differs from application to application.
- Some applications use webhooks for the account/org level, while some use it at a resource level and others use both levels. You will want to host webhook handlers for each application, user & resource (Ex: Contacts, employees, and opportunities)

Estimated Time:

1 month

Monitoring & Alerting

Integrations are not “build once and be done”. You need to provide continuous support post-release when your customers are creating their first integration in your platform and after the integration is published. You also need to consider your customer SLAs in case failures occur to take corrective measures to properly triage the issue as fast as possible.

Main Considerations:

1. Alerting mechanism

- Alerting your customers as soon as a transaction fails is very important.
- The reason for the failure may be the issue with the integration setup or the data. Your customers should be able to know what caused the issue (they will ask).

Imagine if the integration is supporting a mission-critical process such as creating a sales opportunity or sending an invoice to a customer. You will want to be able to triage these issues fast to meet your customer SLAs.

2. Automatic Retry

- The integration should be able to automatically retry in case of network issues or API limit issues. Depending on the third-party API you might want to do exponential or linear retry.

3. Debugging and/or logging system

- Transaction logging is required for both your customers and your support team to quickly resolve the issue.
- When the transaction data is logged you should consider data masking for security compliance. In some cases, you might want the customer to configure which data fields to mask.

Estimated Time:

2-3 months

Testing & Debugging

Testing and debugging features help your customers and/or internal teams set up integrations in your product on their own. This in turn reduces the time it takes to respond to the end user's integration requests.

Main Considerations:

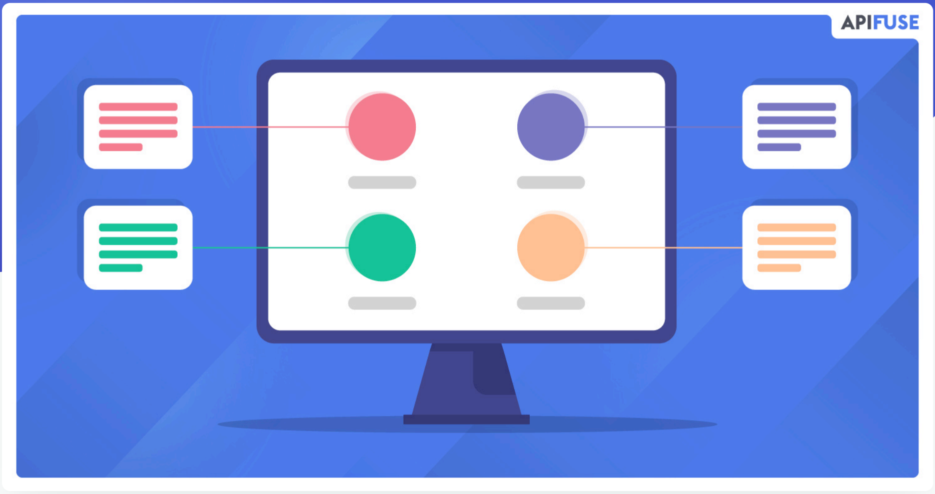
- Your customers should be able to test each step when they are setting up the integration. For example, if they are setting up the trigger for the integration they should be able to test and verify their setup step by step (assuming it is a multi-step workflow).

Estimated Time:

1 month

Orchestration:

Point to Point integration creates multiple bottlenecks as the more integrations you build will often lead to **"spaghetti code"** which gets difficult to manage and maintain. Application orchestration provides an approach to integration that decouples applications from one another. This enables you to route messages, transform events, and most importantly, it provides a way to manage and monitor your integrations from a central location.



Main Considerations:

1. Individual applications/API calls/transformation logic as triggers, actions, or steps

- When you build point-to-point integration you may have everything in a single module like Source/Target API Calls, data transformation, and data mapping. But if you want to build reusable integration components and reduce the complexity you have to split the integration into smaller modules like triggers, actions and events.

- Actions can be anything like making API calls, transforming request/response payloads, executing custom scripts, etc.

2. Decoupling event flow and processing

- To build a fault-tolerant and scalable orchestration layer you must decouple event flow and event processing.
- You can use event buses or message queues or topics for decoupling events and the processing.
- When you decouple event flow and event processing you are building an orchestration layer.

3. Orchestration requires a special font-end that we call the “workflow builder” which can be drag and drop based or guided configuration setup. Which itself requires multiple back-end/front-end components.

Estimated Time:

2-3 months

Integration No Code User Interface (Builder)

The no-code integration builder is a front-end component that enables your customers or internal business team to create and publish integrations. This is a critical component for maximizing integration usage (as mentioned earlier, each customer will have different requirements). Depending upon whether you build a point-to-point or application orchestration integration solution, your workflow builder's complexity may increase.
For point-to-point integrations, the integration builder will be simpler. It can be simple wizards with mostly 2 – 3 steps.
For application orchestration, the integration builder will be complex. You may want the user to drag and drop each step/action and configure the steps on separate popup forms.

Main Considerations:

1. Dynamic Configuration Form/UI

- Whether it is a point-to-point integration or application orchestration, you need to get the inputs for each application from the user to invoke the third-party API. For Example, to send the data into Google sheet you need the drive, file name, and sheet information. These inputs will change application to application and API to API. So your front-end components should be able to build a dynamic form depending on the application or how the action is configured.

2. Input validation

- Depending on the application your input validation can be static or dynamic. Hence, you might want to build a special back-end API that validates the dynamic input forms based on the application.

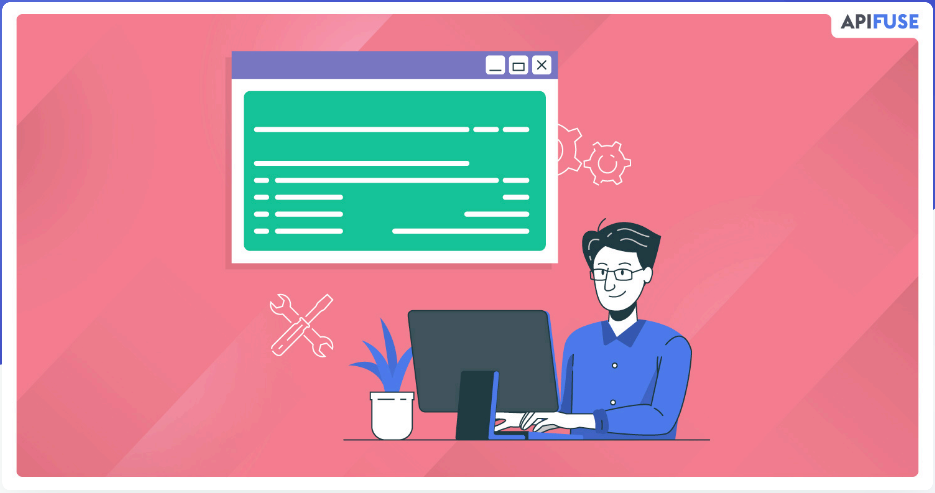
3. Data flow representation in the UI

Estimated Time:

9-18 months

Transformation:

As mentioned, if you want the ability for your end-users to modify the data mapping between the source and target applications this will require additional UI and back-end components your product and development team must consider.



Main Considerations:

1. Formulas

- You must build a suite of formulas for each data type.

2. Bulk data mapping

- Bulk data mapping is required when the API operation has any number of fields that need to get mapped. We firmly believe this is a critical requirement as every customer’s internal systems are setup differently.

3. Arrays and Objects

- API requests/responses may have complex structures like arrays and objects. You will need additional front end components (in the workflow builder and formula editor) to identify the parent object or child object, extract the required field (or array) and map it to the end destination field.

Estimated Time:

1 month