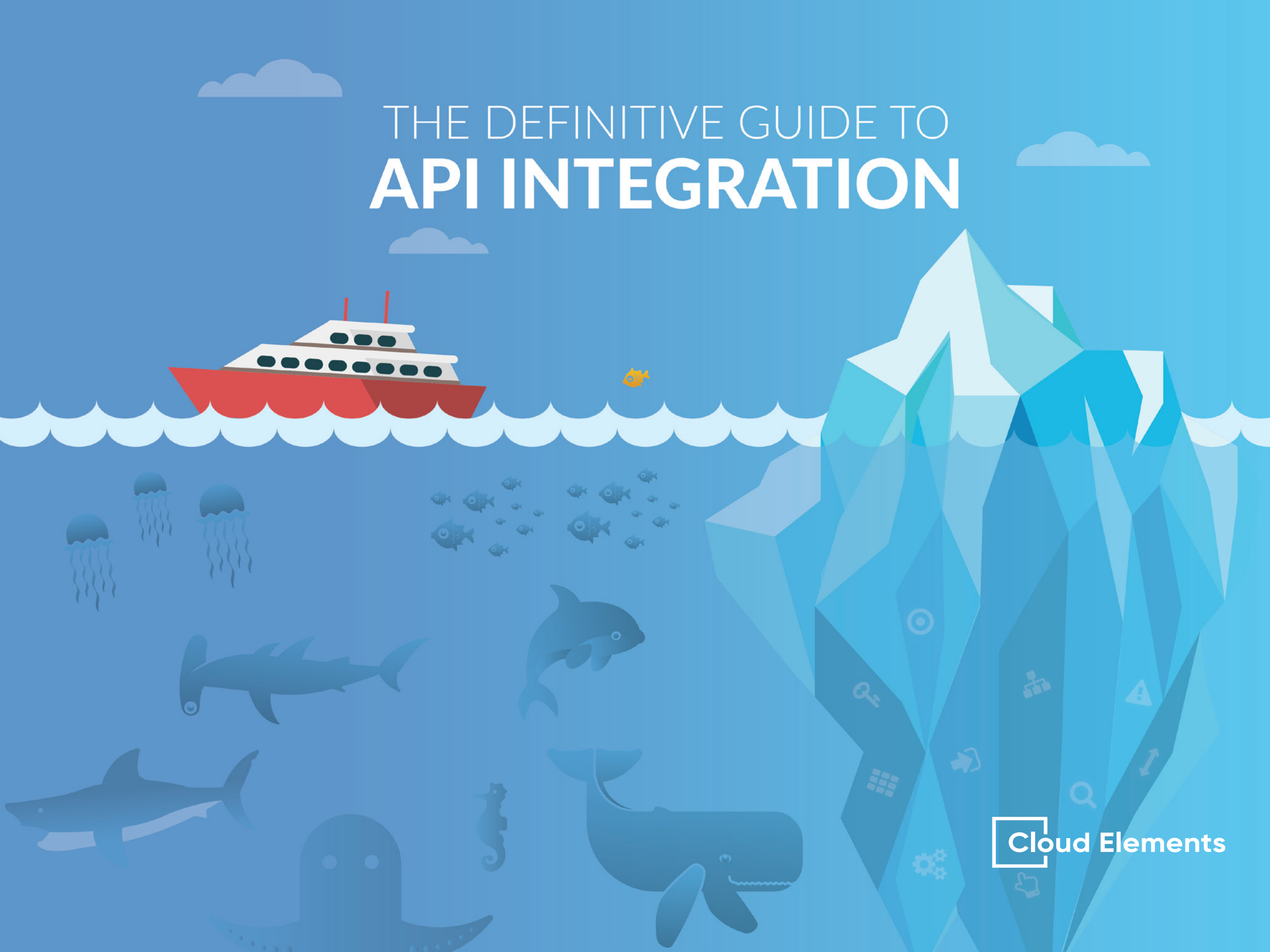
THE DEFINITIVE GUIDE TO
API INTEGRATION



Cloud Elements

TABLE OF CONTENTS

INTRODUCTION

THE DEMAND FOR API INTEGRATION IS ACCELERATING.

Trick is, building integrations one at a time takes too long and costs too much. Your developers are superheroes, but that doesn't mean we have to expect them to have super powers. When it comes to API integration, there is so much below the surface that most of us don't think about on a day-by-day case.

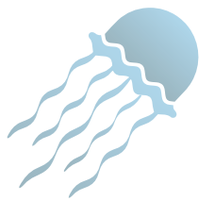
Let's dive in.

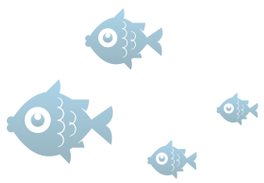
SECTION 01

DEFINING INTEGRATION USER STORIES AND USE CASES

INTEGRATION IS A MEANS TO AN END; A CONNECTED EXPERIENCE WHERE DATA FLOWS SEAMLESSLY BETWEEN YOUR APP AND THE OTHER APPS YOUR CUSTOMERS ARE USING.

In order to build the best user experience, pre-integration work *(defining user stories, knowing your use cases, and selecting integration endpoints)* is a critical.



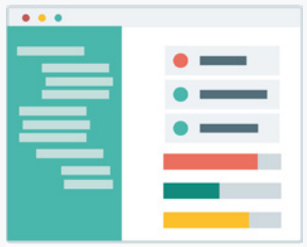


INTEGRATION USER STORIES

Product managers are responsible for deciding on which integrations to build into their application to create a seamless experience with the services and applications used by their customers. To guide the development of these integrations, it's helpful to write integration user stories that provide a clear picture of the experience you envision for your end-customers.

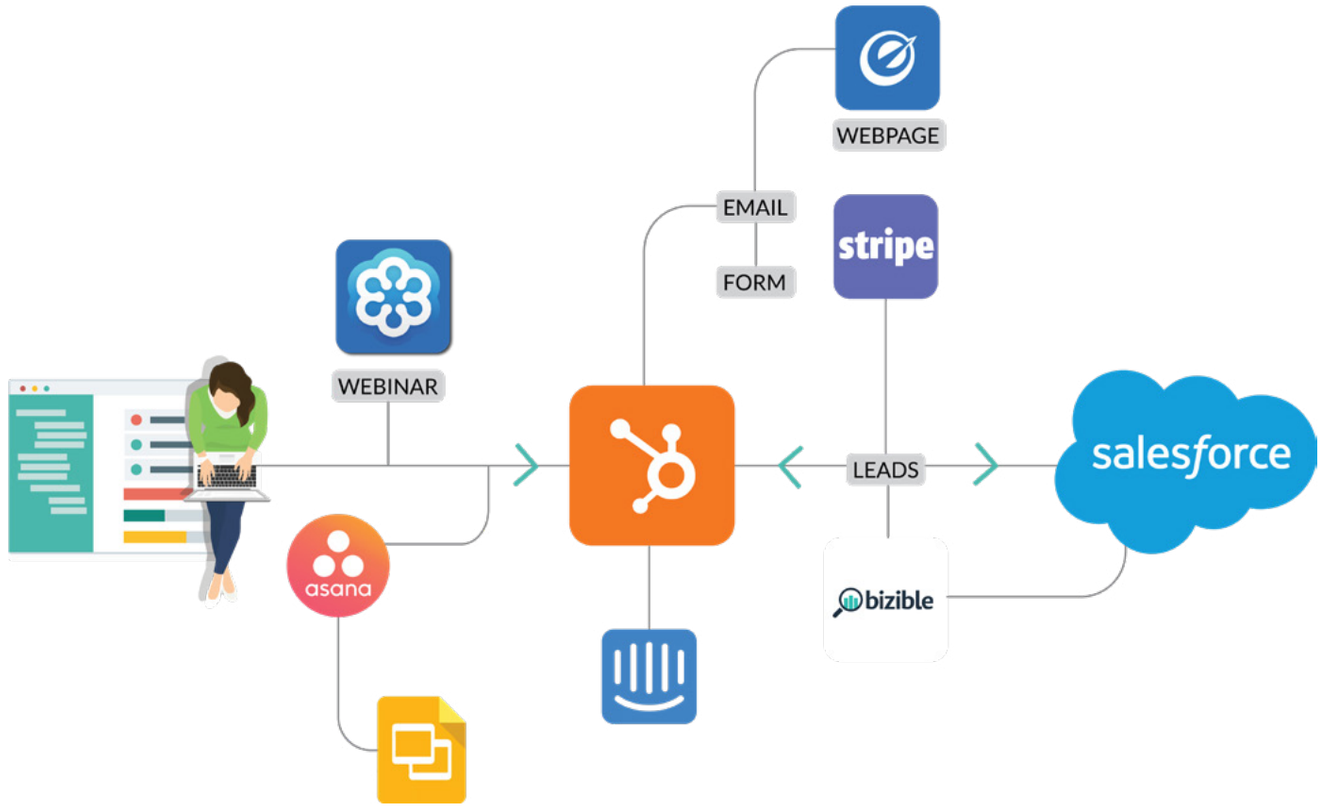
User stories can be defined from two different perspectives: a system persona or a user persona. We recommend user personas, because they typically provide more context and focus on the customer experience.

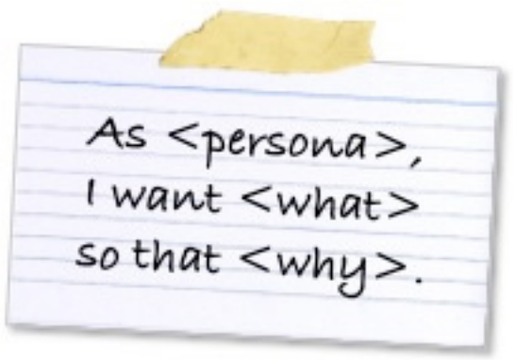
Let's use Mary Marketing as an administrator user example. A Marketing Administrator selects and authenticates their marketing automation application (e.g., HubSpot, Marketo) so that leads are automatically created from the email addresses of webinar attendees. This is a one-time setup that doesn't need to be in the direct flow of the application for the end-users - a 'set it and forget it' function.

SYSTEM PERSONA	RECOMMENDED: USER PERSONA
By developing user stories in the system persona, Product Managers can become too focused on the technical aspects of the integration instead of the collaborative user experience. System based integration stories don't enable the Product Manager to fully visualize the workflow that the user will experience; they guide the integration as a technical solution but not as effectively as a business solution.	A great user persona is written from the point-of-view of an end user. That is contrary to a system persona, designed to specify technical interactions between systems. It's important to get as much use out of these personas as possible to ensure that your stories relate back to an experience model from the viewpoint of your end users.
STORY	**STORY**
The Webinar Management Application will automatically create leads and associate them with accounts in the selected Marketing Automation System so that the two systems will stay in-sync. 	 As Mary Marketing, I can select the Marketing Automation Service that I would like to connect to my Webinar Application so that I can create leads from webinar attendees.

KNOW YOUR USE CASE

In this example, and as a theme throughout the rest of the eBook, we will be discussing API Integration design for SaaS (cloud-based) applications to other cloud services. This is an external integration use case, app-to-app.





Follow these steps to develop your own integration user stories:

01. Start by developing stories as a user persona whenever possible to provide the clearest perspective on how and why your users will benefit from the integration. Capture the user experience through these stories.

02. Refer to the target and source systems that you're integrating to clarify how the data is moving between your application and the service(s) you're integrating and cooperating with.

03. Only write user stories with a system persona when you need to elaborate on the behind the scenes integration tasks that don't involve users.

SECTION 02

SELECTING API ENDPOINTS

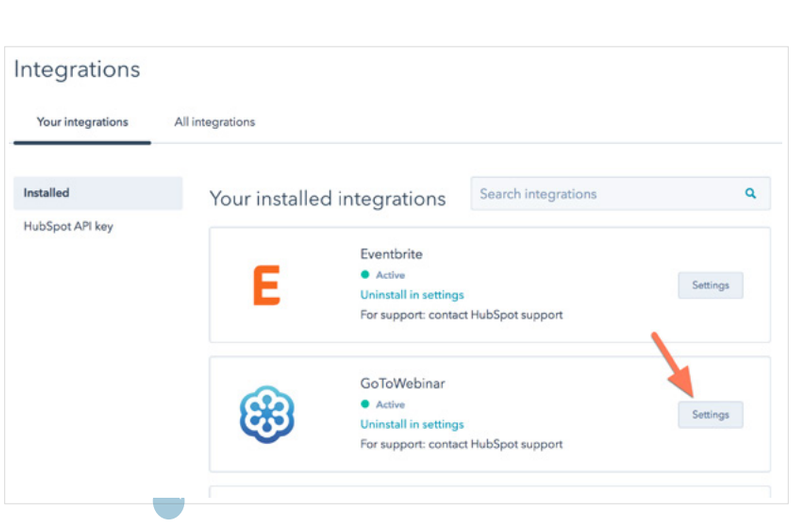
SELECTING API ENDPOINTS

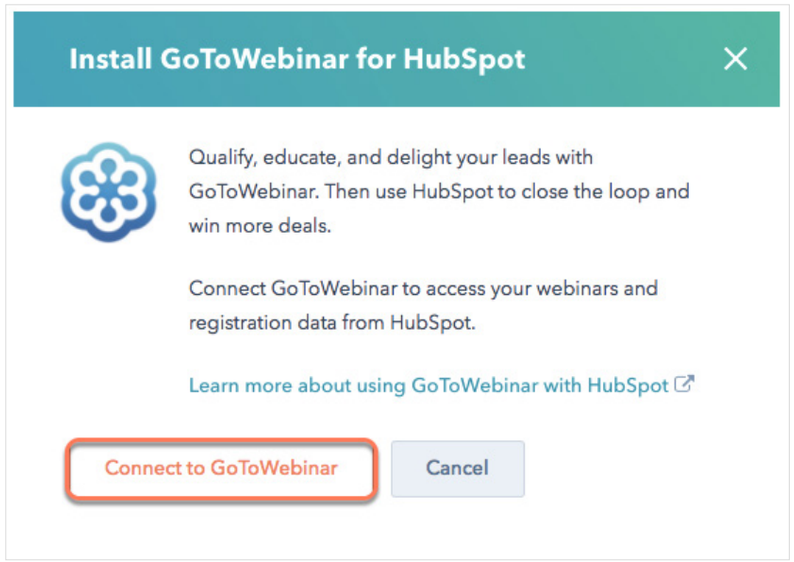
Next, you will want to think about optimizing your app by **selecting the API endpoints** and data needed to integrate with your app. Your goal is to provide a natural experience for your users to setup and authenticate an integration.

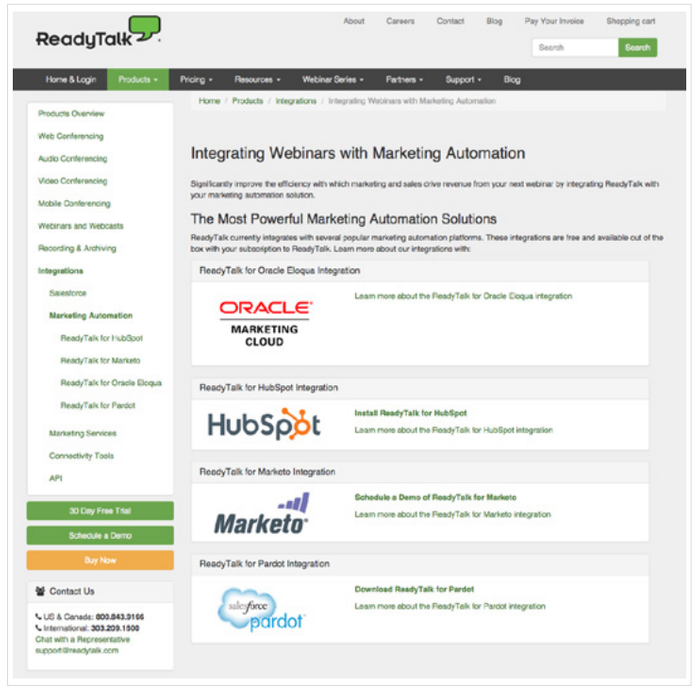
Start by guiding end-users within the natural workflow:

- Mockup a simple selection and authentication flow to determine where it best fits within your app.
- Determine if this a one time event.
- Talk with UX about the best option for your application: Will there be a pop-up window, full-window, or redirect?

In the HubSpot example, the integration end-points are selected by marketing or office administrators. When setting up the integrations, the app admins (or end users) can map their data objects from HubSpot to a marketing webinar platform, like GoToWebinar.







Speaking of data, you want your end users to do as little work as possible when activating an integration, it is best to pre-map the data that your application will exchange with the endpoint so that your users only need to account for custom fields or fields that are specific to that user's endpoint. Your customers should only have to validate and slightly modify data mappings to suit their individual needs, don't make them start from scratch which is prone to error frustration - your app should already have the base fields locked in.

The best integration experience makes end users forget they're even connecting your app with another service. End-point selection and authentication should happen within your application with your brand requesting access to the end-point. You manage the experience for the user from end-to-end.

Don't let your users leave your app!

The last piece to selecting API endpoints is to **anticipate future integration needs**. Many mature apps provide dozens, if not hundreds, of end-points as options for their users to connect to. As you design your user experience, take the time to get to know your customers and learn what matters most to them.

Ask yourself these questions to anticipate future integration needs:

- Which categories of services do my customers want my app to connect to?
- What end-points do my customers use regularly within each of these categories?
- How would you quantify the value of this connected environment?
- What value do these integration points have on the user experience?

Now, you are ready to dive deeper and integrate!

DIVE IN:

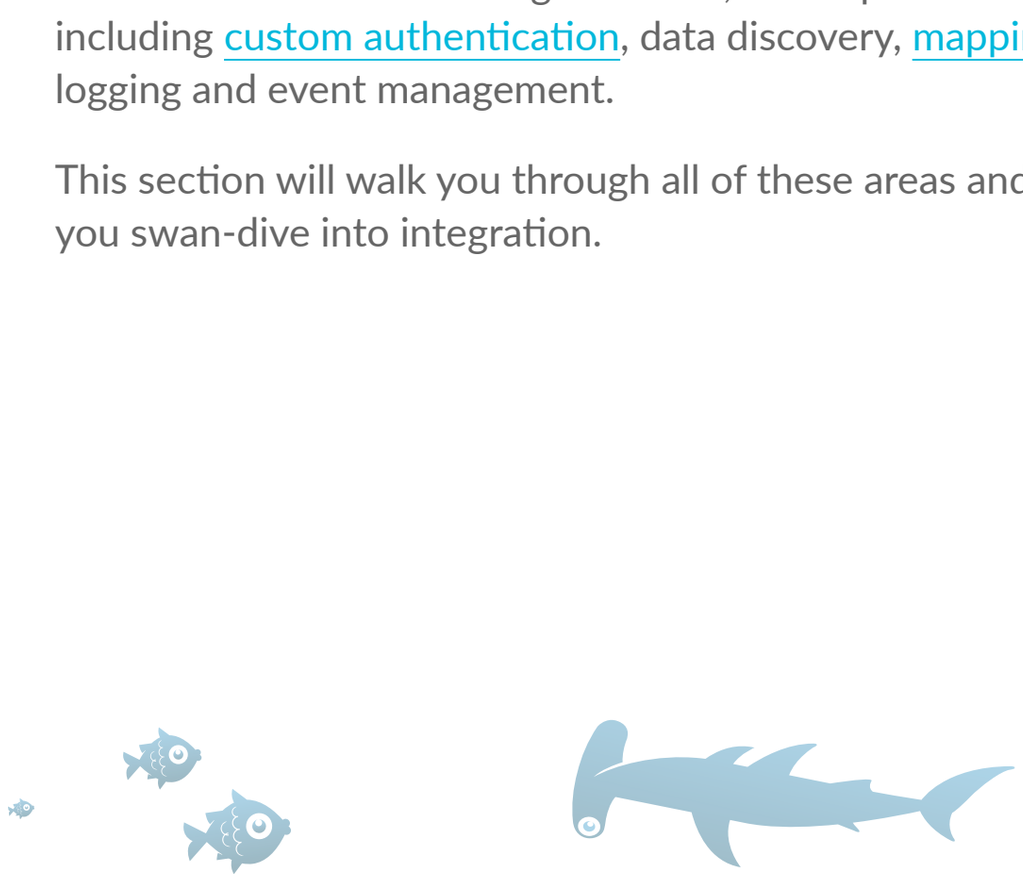
LET'S INTEGRATE

SUCCESS!

Pre-integration work is complete! But wait… Once you are ready to integrate, there are initially more questions than answers. API Integration is about building pre-built integration into your application. However, it's more complex than point-to-point connection, and it's bigger than simply writing directly to the API.

Below the surface of writing to an API, developers are faced with complexities including custom authentication, data discovery, mapping and transformation, logging and event management.

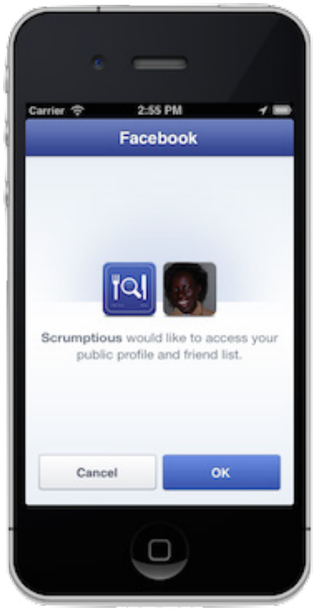
This section will walk you through all of these areas and break down steps to help you swan-dive into integration.



SECTION 03

IDENTIFYING THE API AUTHENTICATION MECHANISM

AUTHENTICATION



Securely and reliably **authenticating** to the cloud service endpoints that connect with your app is the foundation for your API integration experience. You'll need to determine the type of authentication mechanism used by the endpoint and the workflow required.

To get started, ask yourself these questions:

- Which persona will be authenticating to the service? A system administrator or the individual end-users?
- What are the types of authentication mechanisms supported by each service I'm connecting my app with?
- Are there specific security concerns based on the type of data that I am enabling my user to access?
- Where should the data for logins, passwords, keys and refresh keys be securely stored?

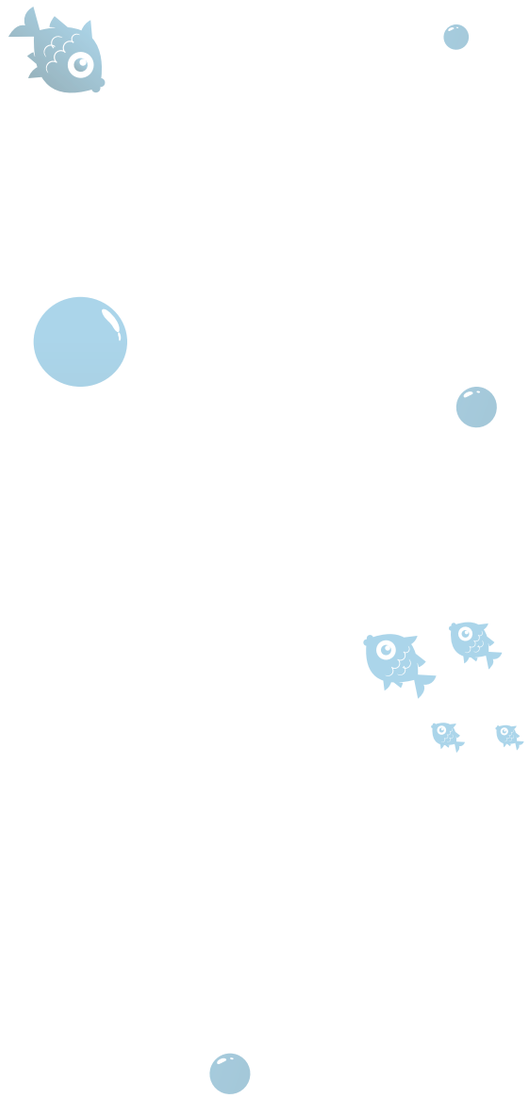
Once a user is presented with the UI to select the endpoint they wish to connect to, your application will open an authentication screen. Users will use their own login credentials to access the service and grant access to connect the apps.

Start by **identifying the authentication mechanism**. The authentication mechanism drives the workflow for each instance an endpoint connects to your app. There are four primary types of authentication mechanisms you can use to connect your app to a cloud service endpoint: Basic Authentication, OAuth, OAuth 2, and SAML.

THE FOUR PRIMARY TYPES OF AUTHENTICATION USED AT OPEN API ENDPOINTS:

- Basic Auth - (or Basic access authentication) is a widely used protocol for simple username/password authentication. This type of mechanism provides no confidentiality protection for the transmitted credentials.
- OAuth – An Open Protocol that provides a process for end-users to authorize third-party access to their server resources without sharing their credentials using user-agent redirections. Credential tokens are long lived, typically a year.
- OAuth 2 – Delegates security to the HTTPS protocol. OAuth 1 does not require this and uses alternative methods to remain secure. It also introduced the use of refresh tokens that allow authentications to expire, unless "refreshed" on a periodic basis.
- SAML – An XML-based open standard data format for exchanging authentication and authorization data between parties, in particular, between an identity provider and a service provider. SAML is popular with large single sign on organizations, corporate applications and in many older applications.

MANAGE REFRESH TOKENS



Next, determine how to **manage refresh tokens** and make sure they are securely stored. If you want your user to authenticate once and then not have to authenticate again as they interact with the endpoint, your application will need to manage the refresh token from the endpoint (if available) in addition to the initial access token. Some access tokens expire in an hour, while others may last as long as a year or never expire. (See a related blog for details on the duration of cloud storage refresh tokens).

QUICK TIPS ON REFRESH TOKENS:

- OAuth 2 is the only mechanism that has refresh tokens. This allows you to keep the token alive, so that users do not have to log in again. SAML does not use refresh tokens. Services like Xero, Quickbooks, etc. do not yet support OAuth 2.0.
- Refresh tokens have similar variations in how long they are authorized for access. You will need to develop a strategy for managing access and refresh tokens for each endpoint that you connect to in order to provide a seamless user experience for your clients.

Tokens are storing very private user information. As a developer for the integration between your application and an endpoint, it's your responsibility to protect this type of sensitive information. If your application stores tokens it is highly recommended that you encrypt with 256 bit encryption at rest within your data storage system with the key owned by the end user. Encrypted tokens stored with 256 bit encryptions are really tough to break, protecting your application and your customers usernames and passwords.

Once you have a strategy for managing access and refresh tokens, you will need to figure out how to **manage permissions from the endpoint.** This all depends on how that endpoint is structured. As you're designing your API integration, consider whether or not your client will be authenticating an administrator account or an end-user account. How will you handle and pass permissions from the endpoint? An admin account at the endpoint will allow you to manage objects and data across your organization, while protecting client data with specified user's permissions.

Lastly, you will want to **track and log activity for each instance** for support, monitoring and security purposes. When you embed integrations into your application, your customers will expect you to support the integration. Your support team will need access to API call activity logs, error reports and alerts to provide the success experience for your customers. Tagging the API calls with an identifier for each customer's instance of an integration will make it easier for your team to manage and support the specific user.

DATA DISCOVERY

When integrating to application endpoints such as CRM, Marketing Automation or Finance systems, your app will need to **discover** the standard and custom data objects and fields used by your customer's instance of the endpoint. This can be done by following a 3-step process.

STEP 1:

IDENTIFY RELEVANT DATA OBJECTS

Identify which of your application's data objects are relevant to the data you're targeting at the endpoint(s). In most cases, you will only be integrating a subset of the data objects and fields from the endpoint. Your application's data model will provide the basis for determining the data you're interested in integrating with. You will then identify the standard objects at the endpoint that you will be integrating to your app.

STEP 2:

DISCOVER STANDARD & CUSTOM OBJECTS

Your integration will likely need an automated means to discover custom objects and fields at the endpoint if you plan to support the integration and mapping of custom data. Since the majority of SaaS application endpoints make extensive use of custom data fields and objects, you will likely need to discover the data structure at the specific instance of the endpoint in addition to understanding the standard data objects. A standard object will persist across multiple instances of an endpoint, but custom objects and data fields will only exist for a specific instance of an endpoint.

Currently only a small subset of endpoints provide discovery APIs to automate the discovery of objects and their metadata. Plan to use these services when they are available at the endpoint. When data discovery services are not available you can discover the data by doing a GET by ID to retrieve a sample of the data structure at the endpoint.

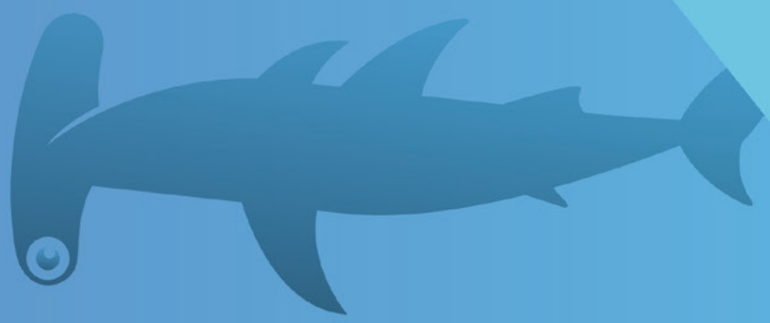
STEP 3:

START TO STANDARDIZE ON PATTERNS

The beauty of taking the extra effort to discover data objects your app is creating is that once you've collected enough data and evidence, you can start to figure out patterns your customers create. Those patterns are what you can use to automate the mapping and transformation process. From those patterns you can define additional fields or objects that your application can potentially use to enhance the integration experience for your customers. For example, you can create a pre-built "custom" object for your app that's installed when clients integrate to a particular endpoint or category of services (e.g., Marketing Automation) that accommodates the typical data fields you find your customers creating when they integrate to those endpoints.

SECTION 04

DATA MAPPING & TRANSFORMATION



DATA MAPPING & TRANSFORMATION

CONSIDER WHETHER YOUR INTEGRATION USE CASE IS STATIC OR DYNAMIC.

- Static integrations provide a fixed mapping between data fields and do not discover custom data and do not provide the end-user with a facility to change mappings. Static integrations are suitable for services with fixed payloads such as cloud storage, payment and messaging services.
- Dynamic integrations are designed to accommodate custom data. These integrations require services to discover custom data, the ability to map data and custom fields for each instance of an integration and often the ability to transform data values between endpoints. They're "dynamic" in the sense that they enable each integration to be uniquely designed based on the data discovered at the endpoint.

If you're providing a **Dynamic** integration experience for your users, you'll want to consider how your users will be able to map data fields and transform data values. With a UI for data mapping, you will enhance the experience to seamlessly synchronize applications.

We've broken down the process to setting up mapping and transformations into five easy steps:

01

Define a Default Map

A default mapping is a template that will define how standard data structures from an endpoint will map into your applications' data model. This mapping establishes the association of standard fields within a standard object into the fields within your application. The primary goal here is to save your users' time by pre-mapping standard data from the endpoint reducing the effort they need to undertake when connecting to your app.

02

Present Discovered Data

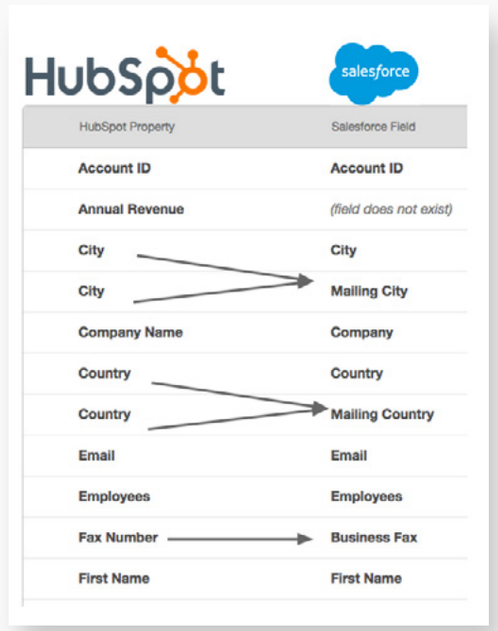
Once your app Discovers the data structure at your customers endpoint you will need to present the discovered structure, custom objects and custom fields for your users' to evaluate and map based on their understanding of the data.

03

Map Custom Data from Each Instance

Custom data objects and custom fields within standard objects are a common feature of most SaaS applications. Invariably your customers will make extensive use of these capabilities. Therefore default mapping will only be the beginning for a majority of your customers. The challenge here is that you're going to need to support unique data maps at the "Instance" level. The instance level data maps will be specific to that authenticated account of an endpoint.

There is no way to anticipate the type of data coming in, as it only exists specific to that "instance" of a service. Custom mappings can be done by an administrator or an end-user at your customer with a User Interface embedded within your app. The most dynamic user experience is to provide them full self-service control over these mappings, as their data structures will change regularly and it will reduce time and cost for you and your customer to make them self-sufficient.



We've broken down the process to setting up mapping and transformations into five easy steps:

04	**Make your Default a "Guideline" - Not a Requirement** There is no way to always anticipate how your users may use standard fields. Occasionally, users will use standard fields for a custom purpose and they will likely want to map this data to a field other than your default mapping. Just as you handle Custom Data on a dynamic self-service basis, it's best to enable your users to override your default settings.
05	**Consider Embedding Self-Service Data Mapping Tools** As discussed previously the optimal experience for your users is to have a user interface integrated with your app to facilitate the mapping of data from their cloud service apps into your data structure. By providing this facility, your customers can make their own mapping using their best judgement and respond instantly to accommodating new data fields and objects. Mapping data is one piece. You have invested time and money into configuring the data to flow into your app in a very specific way. That doesn't mean that the other apps that are on your integration roadmap have the data formatted in the same manner as you do. The next important step is **data transformation**. This is the processing of data from one format, such as XML or SOAP, or even currencies and time formats, into another.

HERE ARE SOME KEY CONSIDERATIONS FOR DATA TRANSFORMATIONS:

- Evaluate each endpoint and determine if values are consistent with the values your application expects. For example, help desk ticketing systems use different priority schemes (High, Medium, Low vs. 1, 2, 3) and you will want to consider if you will transform these values into a consistent data set required by your application.
- Understand each individual payload structure in order to connect to the different endpoints. Cloud services and technologies deal with a variety of payloads, including XML, SOAP, JSON, etc, which means that there will be a variety of formats coming in with the different data objects.
- You can't anticipate all of the transformation needs for each endpoint and each customer. Consider providing a programmatic facility to incorporate custom logic into your transformations.

SECTION 05

API DESIGN: IMPLEMENTING SEARCH, BROWSE AND SELECT

API DESIGN: IMPLEMENTING SEARCH, BROWSE AND SELECT

Once you've mapped the data and ensured the values can be transformed, the next step is to present the user with a way to search and/or **browse** the data that they wish to access.

A few example user stories for search APIs are:

01. As a cloud document storage user, I want to browse by cloud folders and find my documents, so that I can load them into the app integrated to that service.

02. As a CRM user, I want to browse customer accounts by company name, so that I can easily sort and find the right account to sync with the integrated app.

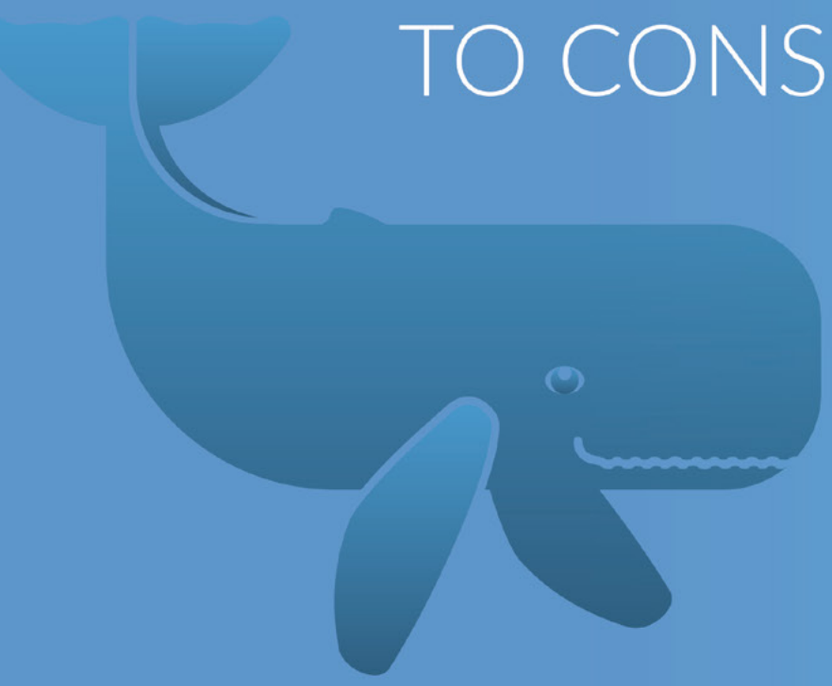
03. As a Finance user, I want to search for paying and non-paying customers, so that our app can stay in sync with my reports.

There are a few areas to consider when planning and defining the API design to have search and browse functionality. First, the search functionality provided by APIs from leading SaaS applications varies dramatically and these limitations will impact the type of functionality you can offer across endpoints. It is important to research the search capabilities that are offered first to determine if you will take a least common denominator approach or write specialized search integrations to each endpoint. Due to this variability by endpoint, search will be one of the most custom parts of your integrations and it may change the use case of your integration depending on the service your client is integrating with.

Another area of variability is how to page the results returned from each endpoint. When presenting the search results from the endpoint you will want to implement a common approach to pagination for your users. Paging is a big factor in how users can search and find the data they are looking for.

SECTION 06

CRUD vs REST VERBS: DETERMINING AN APPROACH TO CONSUMING EVENTS



CRUD VS REST VERBS: DETERMINING AN APPROACH TO CONSUMING EVENTS

It is important to determine how you'll consume events from the endpoint(s) you're integrating with. Many endpoints support webhooks that can be used to track create, update and delete actions as they occur to objects notifying your application. Webhooks keep your app up-to-date with changes (events) at the endpoint.

If the endpoint does not support webhooks, then your application will need to poll the endpoint to identify changes as they occur. With polling, you're querying the timestamps on objects at the endpoint to see if actions (create, update or delete) have occurred. Your app becomes responsible for reaching out on a periodic basis based on a frequency that you set such as every minute, hourly or daily.

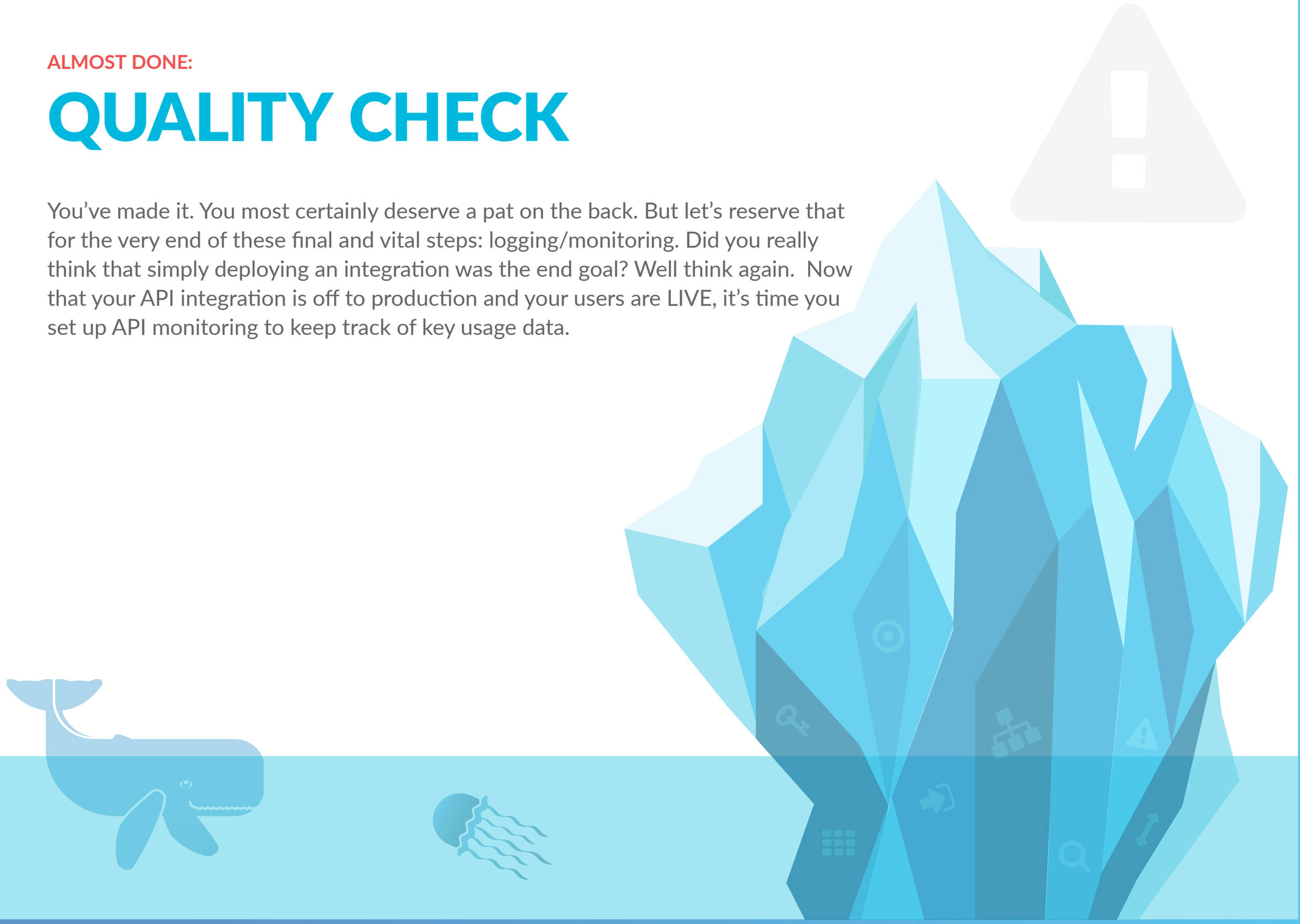
CRUD ACTIONS CREATE EVENTS

Events are critical to giving you the "trigger" to sync data between multiple endpoints. RESTful methods initiate Create, Retrieve, Update and Delete actions to resources at endpoints using GET, POST, PUT, PATCH and DELETE methods. Since the goal is to synchronize data when it's changed, you will need to consume events that Create, Update and Delete data. Consuming events will enable your application to execute corresponding methods to Create, Update or Delete when notified that an event has occurred at the endpoint; thereby synchronizing actions that change data.

ALMOST DONE:

QUALITY CHECK

You've made it. You most certainly deserve a pat on the back. But let's reserve that for the very end of these final and vital steps: logging/monitoring. Did you really think that simply deploying an integration was the end goal? Well think again. Now that your API integration is off to production and your users are LIVE, it's time you set up API monitoring to keep track of key usage data.



SECTION 07

LOGGING AND API MONITORING

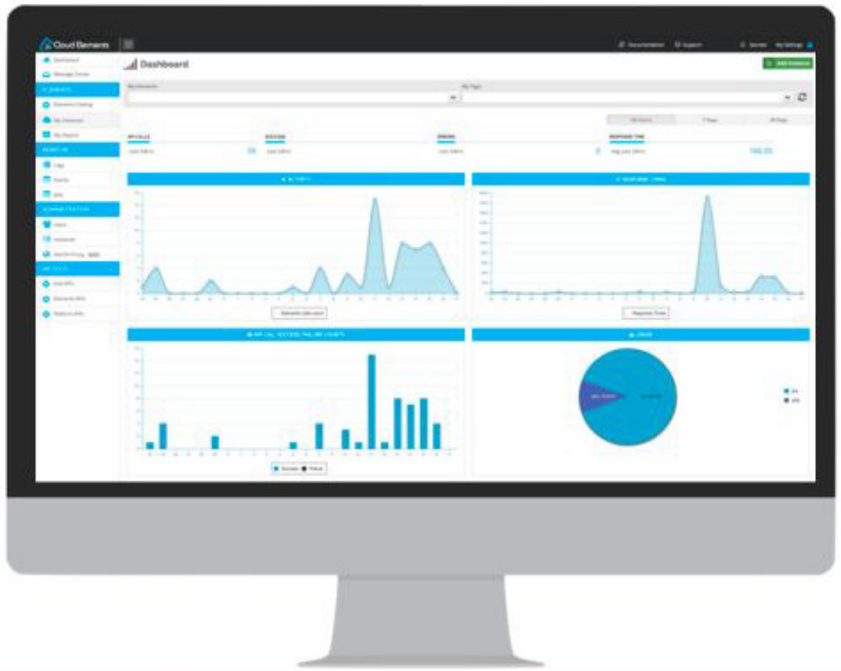
LOGGING AND API MONITORING

Integrations are mission critical. Your customers are depending on your integrations to be up and running, no matter what. We hear over and over that if an app's integrations are not working, customers and end users are frustrated, reflecting poorly on your core application.

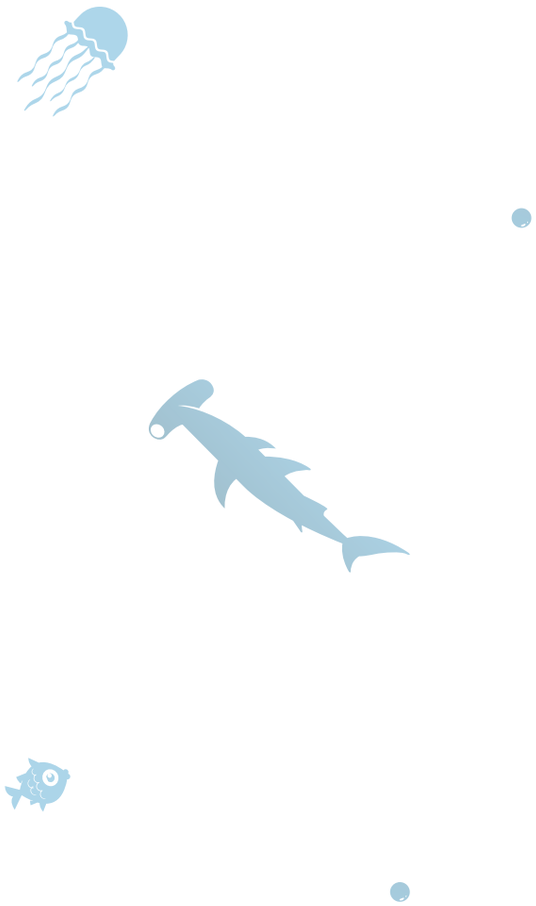
Some of the most common errors to test for include:

01. Authentication credentials change due to changes in passwords
02. API calls are failing
03. The endpoint service may be down completely or purposely shut down for scheduled maintenance

We have compiled years of integration experience into useful advice to help you not only prepare for, but hopefully avoid common integration failure modes.



FINAL CHECKLIST FOR A SUCCESSFUL INTEGRATION



Stay on top of usage data: User experience data is essential to your integration's success. Measure your stats and feed them into your application teams' performance monitoring tools (APM). Monitoring this data enables support to better understand usage patterns and establish proactive support of your client's integrations.

Regular health checks: Frequent endpoint health checks ensure that your app won't get the blame when things go wrong and you can proactively assure customers who may be worried about your product's stability.. We recommend automating health checks for your app's integrations to determine if any issues are arising from the endpoint, your integration, or your app.. A simple ping to the endpoint can diagnose one of the most frequent integration issues.

Track response times: Avoid response time issues by isolating your application's performance vs. the performance of the endpoint you are connecting to. Some endpoints are notoriously slow and you will want to make sure to set proper expectations with your clients for performance delays when integrating to slower endpoints. If slower response times persist, you can also implement a cache or a number of other mechanisms to improve performance.

Tag your usage data: Tagging an integration's usage data enables rapid filtering of your logs to isolate and debug issues. Tags are vital to enriching your reporting and dashboard experience. Use tags to dig into the granular data and fully support filters, searches, and queries.

Standardize error responses: Your operations team will not have time to research every vendor's error code. Standardize the error responses your application currently runs on so that your support team can understand and react to errors more rapidly and efficiently. While we recommend that you normalize error codes, we also advise providing a payload back from the endpoint on how errors are displayed.

Lastly, you will want to **track and log activity for each instance** for support, monitoring and security purposes. When you embed integrations into your application, your customers will expect you to support the integration. Your support team will need access to API call activity logs, error reports and alerts to provide the success experience for your customers. Tagging the API calls with an identifier for each customer's instance of an integration will make it easier for your team to manage and support the specific user.

CLOSING

And there you have it. If you follow these steps to plan, integrate, and test your API integration, you will provide your customers with the absolute best app integration experience. If you're ready to take the plunge into building your own API integration, check out our Cheat Sheets for Endpoint Integrations or view our resource center for other easy to follow information.

If you have questions, or if you would like to learn more about Cloud Elements, contact us today!

ABOUT CLOUD ELEMENTS

Cloud Elements is the leading API integration platform for SaaS companies and the digital enterprise. Integration can be far harder than you think! It's more than simply connecting to an API. Cloud Elements has re-imagined the world of application integration, by allowing you and your customers to focus on the data they care about.

Underneath our catalog of more than 150 Elements, you'll find an entire platform to go beyond just connecting, and instead truly integrate and synchronize data between applications.