API Integration Best Practices

CONTENTS

WRITTEN BY CHASE DOELLING
HEAD OF PRODUCT AND ALLIANCES MARKETING AT CLOUD ELEMENTS

Introduction

Awesome APIs make for even better integrations, but like the APIs they connect, integrations can be unique and vary wildly. Yet, integration is becoming necessary, as many customer experiences are driven by aggregating existing data from applications instead of recreating it. Often times, creating enhanced views can speed up decision making by having your data available in one place. Integrations optimize processes by automating the transfer of data from one application to another or multiple applications, serving not only internal departments, but also customers, prospects, and partners.

The number of applications organizations use to conduct business on a daily basis is not only staggering but also increasing year over year. To use an example, there are now over 7,000 marketing applications to choose from that are being tracked by chiefmartec.com. Now think about all the applications engineering, product, QA, finance, accounting, and support use! Each of these applications will likely have their own APIs and, further, they may have different styles of APIs such as REST, SOAP, SDK, etc. So how do you bring all the data hiding in these applications together?

This Refcard serves as a starting place for some best practices when approaching integration. We'll begin with documentation and understanding the target application's endpoints, before learning to implement authentication to gain access to the data you need. Next, we'll cover eventing APIs and bringing that data to life by creating flows between applications. We will discover how to query for just the data we need, and finally, cover paging through those vast troves of data and moving it in bulk if need be.

Navigating Documentation

Documentation is the aloha of API integration, where it both starts and ends for developers. Understanding what data you are getting from the endpoint provider is a crucial first step, but you also want to replicate documentation best practices if the integration you're building will be used by other team members and shared with other developers. Many popular SaaS applications have easy to navigate documentation, but, in many cases, full descriptions may be lacking to the point where you're not sure what CRUD operations are supported.

Cloud Elements

Unified APIs

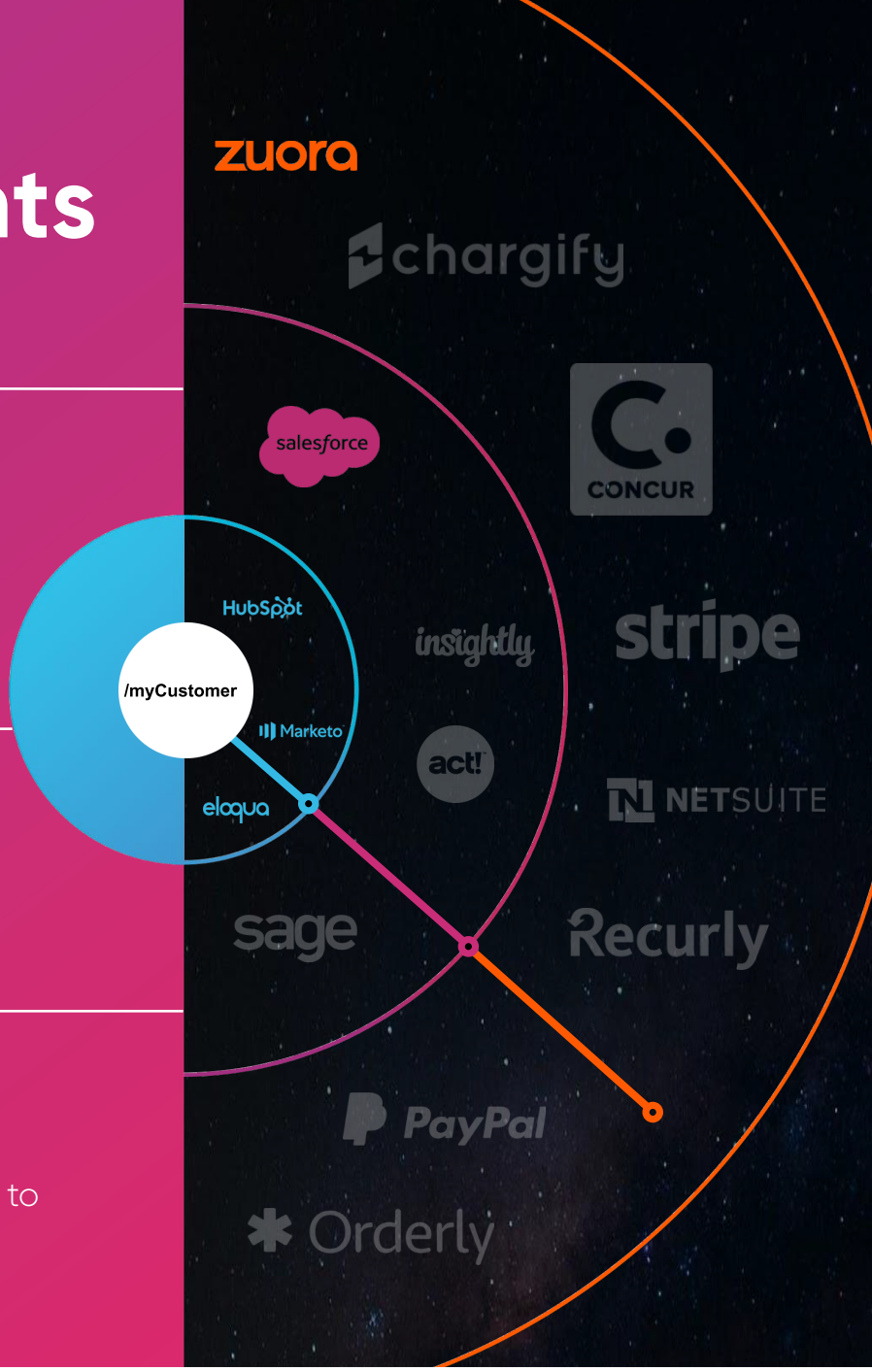
100% REST-based, providing a standardized approach to authorization, eventing, error handling, paging, and searching - regardless of endpoint implementation.

Virtualized Data Hub

A canonicalized view of your data model, mapped across multiple applications.

Formulas

Formulas are workflow template that are independent of the endpoint and are used to define a set of actions that will take place based on an event happening.



OWN YOUR APP ECOSYSTEM

Cloud Elements is the first API Integration Platform to virtualize APIs into unified data models that eliminate brittle, point-to-point connections. Free your developers to discover and build what's next.

LEARN HOW

When working with a new application, try to find a working example that you can build upon to learn the application's APIs. If you can't find one within the API documentation, see if there are any tutorial videos, webinars, or blogs. Learning by example with new applications is the best way to understand if there are nuances, gotchas, workarounds, exceptions, and time-saving tricks that might save you hours of confusion down the line.

Once you're ready to get started, find the *root* of the API as the base and then pick a GET call on a resource you feel familiar with; typically anything with contact data (names, places) is a great place to start. Call the resource using an API testing tool (like Postman) or Curl command. This will allow you to create a baseline of what data structures you should expect and to use as a target for the integration and how it could be formatted in a UI.

Quickly Navigating Endpoint Documentation:

- Make sure you have admin or developer access
- Find the API Root
- Look for a familiar data structure to test against
- Make a test call to use as your baseline

Before we move on it's worth noting that if you're at the other end and creating integration documentation, you should specify how users should interact with your integration and what they should know going in.

Key Pieces for Creating Integration Documentation:

- Introduce the use case the integration solves (ex. Notification of high spending).
- Explain what the integration does, specifically with omitted or added fields (ex. SMS if "Approved Amount" is greater than $10,000).
- Explain the level of access the user needs and how to get it (ex. Normal, Developer, Admin).
- Create an FAQ for anticipated troubleshooting or know limitations (ex. Times out after 15 minutes).

ERROR CODES

When creating integrations, it's important to understand what error codes you might be getting from the application provider. As you start sending requests, it is helpful to understand when things work, but it becomes equally important to understand why they don't. With integrations, error codes resulting from a lack of access to size limitations help guide your application's business logic as well. While many error codes are not normalized here is a general reference guide when working with REST APIs:

CODE	ERROR & DESCRIPTION										
1XX	Informational	200	Executed	302	Temp. Found	401	Unauthorized	410	Gone	415	No Media Type
2XX	Success	201	Created	303	See Other	403	Forbidden	411	Length Req.	417	Failed
3XX	Redirection	202	Accepted	304	Not Modified	404	Not Found	412	Precondition	500	Internal Error
4XX	Client Error	204	No Content	307	Temp. Redirect	405	Not a Method	413	Entity Too Big	501	Not implemented
5XX	Server Error	301	Permanent Move	400	Bad Request	406	Not Accepted	414	URI Too Long	503	Unavailable

AUTHENTICATION

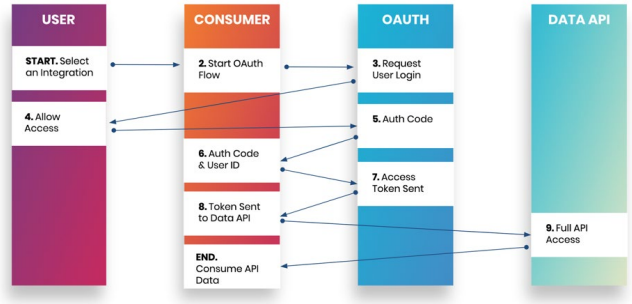
Getting the right access to the right data underpins any integration project, yet authentication can be one of the hardest parts. Authentication, at its core, is the ability to prove your application's identity. There are several different ways that applications can grant access to developers to create integrations.

BASIC CREDENTIALS

This approach is listed for educational purposes, but it is, fortunately, becoming less common due to its security vulnerability to man in the middle attacks. This method uses the username and password in the HTTP header when you are making calls, so the security becomes dependent on the availability of SSL. This method lends itself to internal APIs for resources that aren't generally exposed elsewhere or for IoT projects.

API KEYS

The use of API keys is still pervasive and very common for API access. It is a fast, easy way to authenticate accounts and is relatively low overhead for the provider as the provider can easily create and purge API keys. API keys are a uniquely generated set of characters, sometimes random, and are often sent as a pair, i.e. a user and a secret. When receiving API keys, it's a best practice to copy and store them in a password manager and treat them like any other password.



OAuth is the preferred and best practice approach to authentication because it allows users to decide what amount of access the integrating application can have while also being able to set time-based limits. Some APIs have shorter time limits; in these cases, refresh tokens are required.

EVENTING

We've navigated documentation, authenticated access, and can GET a "Hello World" response, now let's bring the data to life, or at least get it automated. In some cases, eventing isn't needed and you can skip ahead. However, the true power of integration is the ability to put data in motion without the bottleneck of human clicks. There are two primary ways to automate or "event" your API calls.

POLLING

Polling is very much like the kid sitting in the backseat on a road trip constantly asking, "Are we there yet?" Except computers don't get annoyed, so the conversation goes like this:

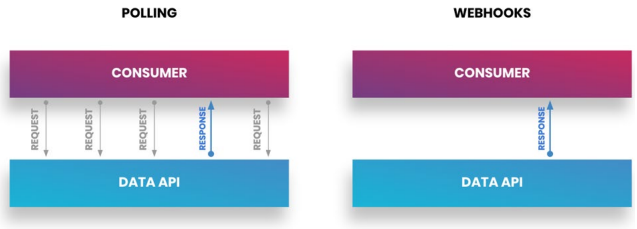
"Any changes in the data yet?"

"No, no, no, no, yes, no, no, yes," and so on.

To detect if there have been any changes in the data (specifically create, retrieve, and delete events) polling is sending requests at a predetermined frequency and waiting for the endpoint to respond. If there is no response body then no changes have occurred. Because of this continual request for information, polling is very inefficient and can be expensive when paying for computing time, as the vast majority of requests go unanswered. Finally, because of the intervals in timing, polling is immediately out of date once the API call returns, batching any new updates that would have happened since the last call.

WEBHOOKS

Webhooks allow you to get updates as they occur in real-time; they are *pull-based* instead of *push*-based. In this method, when updates occur from the application, they are sent to a URL that you've specified for your application to use. This push-based refreshing of information is what gives applications the ability to update in real-time and create dynamic user experiences. Additionally, it is the preferred best practice as the implementation is just a URL. This is opposed to polling, in which you need to create a polling framework that manages the frequency and scope of the information you wish to receive. To give an example, Marketo's polling framework is 239 lines of code compared to just 5 lines for a similar webhook.



QUERYING

We've automated the movement of data, so our integration is live and fluid, but now it's flowing in like a river when all we really need is a small piece of the data that is useful to what our application is doing. We need to set *query parameters* to filter out all the data we don't need. Querying is the last part of the API endpoint, or path, that directs the call to pull only the data you want.

```
GET /widgets***?query\_parameter\_here***
```

The ability to search against databases as well as the target applications is a crucial step that allows you to test and confirm responses and see any difference in the data structure between what is returned and what is in the documentation. Querying allows you to modify the key-value pairs of the request. At the end of the URL path, the query starts with a (**?**) and each query is separated with an ampersand (**&**). For example,

```
GET /widgets***?type=best&date=yesterday***
```

To test an endpoint, there are multiple options depending on which language you prefer, but the majority of API documentation will reference commands in *curl*. "cURL" stands for Client URL and is a command line tool that becomes particularly valuable when retrieving data from other applications. To test an API, open up your terminal and copy-paste the curl command to see what response you get back. Simple enough right? Now try adjusting the key-value pairs and fine tune to just the information your application needs from the target endpoint. When adding parameters to a curl command directly, be sure to add a backslash (/) before the ? of the query as (?) and (&) are special characters.

Tweaking these pairs will decrease the size and overhead your application might expect. Simplicity is the act demystifying something without losing the magic, and the same could be said for APIs. You can always increase the amount of data (especially in testing) to see where you might run into errors.

SORTING AND ORDERING

An important piece to getting to the information you need is how the data will be presented at first glance, which becomes especially helpful in testing and presenting data through a UI. Many APIs will support *sort*, *sort_by,* or *order* parameters in the URL to change the ascending or descending nature of the data you're working with. For example, the following can give us the freshest widgets in our inventory:

```
GET /widgets?sort_by=desc(created)
```

PAGINATION

There might be way too much data or data that just keeps going and scrolling. This is where pagination comes in. Pagination is the ability to put that giant response of every customer you've had since 1994 into human readable pages without seizing up your computer. Pagination requires some implied order, like a unique ID, Date Created, Date Modified, and so on. There are a few different types of pagination you might encounter.

OFFSET

This is the easiest to implement and is, therefore, very common. The *LIMIT* is the number of rows that will be returned and the *OFFSET* is the starting point of results. For example, a path of `/widgets?limit=10` would return 10 rows for the first page. Then the second page could be `/widgets?limit=10&offset=10` would return 10 rows, starting with the tenth row, and so on. The downside of using offset pagination is it becomes less performant with large data sets. For example, if the offset is 1M rows, it will still have to run through 1M rows before returning the limit requested.

KEYSET

A keyset is helpful to get around large data sets and uses the filter value of the last page of results as the starting point for the next page of results using an additional limit URL parameter.

For example, the first call could be `GET /widgets?limit=10` with a unique identifier such as date created or modified. The second call would be `GET /widgets?limit=10&created:lte:2019-09`. The next call would be `GET /widgets?limit=10&created:lte:2019-10` and so on.

FIXED DATA PAGES

This is fairly straight forward. To add fixed data pages into the query, you select which page of data you would like returned. For example, using `GET /widgets?page=4` we see the fourth page of results are returned. This method is preferred if the application with which you're integrating has known pages and you would like the data ordered sequentially. It becomes harder, however, if you're not sure what exactly is on that fourth page.

FLEXIBLE DATA PAGES

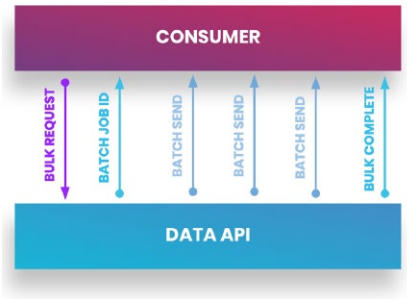
Similar to fixed data pages, you could still call the fourth page of widgets but now you can specify the size of the page. Calling `GET /widgets?page=4&page_size=25` allows you to dictate even further what you will get back in terms of size of the page. This can be very helpful if you're building a UI and need the results to be a certain size.

BULK

Up to this point, API integration has focused on specific sets of data, but what happens when you need to move a large amount of data from one system to the next? Some applications expose this ability with Bulk APIs. Bulk APIs allow you to update, insert, and delete a large number of records all at once. This can be particularly useful when transferring large systems of record (marketing automation, CRM, or ERP) from one provider to another.

Bulk actually operates in several *batches* of data sets when a request for a bulk job is sent to the application provider. The application provider will send batches over asynchronously to complete the job. Depending on the size of the data set the job will also send you a unique identifier to check the job status and close the job once complete. Be sure to double check the file type that a bulk API will provide, either CSV, JSON, or XML. Additionally, if you're not getting the full data set back, be sure to check any API rate limits that apply, as you may exceed them with large data transfers.

BULK DATA



Resources

BOOKS (From the REST Architecture Card)

- "RESTful Web APIs" by Leonard Richardson, Mike Amundsen and Sam Ruby, 2013. O'Reilly Media.
- "RESTful Web Services Cookbook" by Subbu Allamaraju, 2010. O'Reilly Media.
- "REST API Design Rulebook: Designing Consistent RESTful Web Service Interfaces" by Mark Masse, 2011. O'Reilly Media.
- "RESTful API Design" by Matthias Biehl, 2016. CreateSpace Publishing.
- "REST in Practice" by Jim Webber, Savas Parastatidis and Ian Robinson, 2010. O'Reilly Media.
- "Restlet in Action" by Jerome Louvel, Thierry Templier, and Thierry Boileau, 2012. Manning Publications.
- "Developing RESTful Services with JAX-RS 2.0, WebSockets, and JSON" by Masoud Kalali, 2013. Packt Publishing.

Written by **Chase Doelling,** *Head of Product and Alliances Marketing at Cloud Elements*

Chase Doelling is the Head of Product and Alliances Marketing at Cloud Elements and sherpas customers to evolve their integration strategy from product decisions to board driven initiatives. He is a frequent speaker and writer on API and integration topics that focus on making the best use, of best practices, in a very real world. He also struggles to explain what he does to his parents.

DZone
A DEVADA MEDIA PROPERTY

DZone communities deliver over 6 million pages each month to more than 3.3 million software developers, architects, and decision makers. DZone offers something for everyone, including news, tutorials, cheat sheets, research guides, feature articles, source code, and more. "DZone is a developer's dream," says PC Magazine.

Devada, Inc.
600 Park Offices Drive
Suite 150
Research Triangle Park, NC

888.678.0399 919.678.0300