{"logo":"API Evangelist"}

API Industry Guide

Essential Building Blocks Across 21 Areas Of A Modern API Life Cycle

Design, DNS, Containers, Virtualization, Deployment, Management, Monitoring, Security, Terms of Service, Privacy, Licensing, Branding, Client, IDE, SDK, Embeddable, Webhooks, Monetization, Plans, Partners, and Evangelism

January 2016

by Kin Lane, the API Evangelist

This is meant to be a field guide to the fast changing world of APIs, providing a checklist of considerations across 21 areas of API operations.

This is a BETA document!

I will be adding links in next version, as well as some other newer areas of my research.

Just The Essentials of A Modern API Life Cycle

For the first time in five years, I have all my research to a point where I feel like it is getting closer to the view of the API space I have in my head. It's not complete. It never will be complete. I just wanted to take a snapshot of this moment in time, and use it as an opportunity push forward my overall awareness of what is going on.

As the API Evangelist I have been keeping an eye on the API space since 2010. I follow the blogs, Twitter and Github accounts of the companies and individuals doing interesting things with APIs. I have done this every week, for the last five years. AS I look through the websites, and developer areas for the leading companies doing APIs, I look for common elements of the services and tools that they provide, and break them down into what I call building blocks.

I organize these common building blocks of API operations, in individual research projects, and publish more details on the companies, individuals, services, and tools I've come across during this process. I tend to focus on the top APIs in the space, and the leading API service providers, who cater to the sector. With my working coming together so nicely, I'm finally able to pull out what I'd consider to be the essential building blocks across the core areas of the aPI space.

This guide is the first look at my research, ignoring the API providers and API service providers I track on, and just look at some of the common building blocks employed by the API providers, which often times are also echoed by the features the API service providers are peddling. This guide is just getting started, I have a lot more links, and other resources I need to include for each of the building blocks, but this is shaping up to be a pretty good first crack at this high level view of the API lifecycle, if I don't say so myself.

Design

For me, API design is so much more than just a dogmatic belief of REST. API design is about stepping back, and deeply considering everything that goes into your API, before you ever get your hands dirty actually building an API. My API design research is about taking the common elements from the design practices of APIs I watch, but also extracted from the API design guides companies like Paypal or the White House are publishing what they preach across their operations.

This API design research is meant to be a guide, and checklist of the essential elements that I see going into API design, from some common sense principles to think about to open formats, as well as some design processes that can help make the whole thing flow much more smoothly. You might be familiar with some of these API design building blocks, but hopefully there are some areas you might not already be tuned into.

- **Best Practices** - What are the best practices for API design. This is about best practices specifically in the world of API design. This should be about having an overarching philosophy, and ethos when it comes to API design, that reflects the technical, and business goals of a company.

    - **Use the Web** - The web brings a lot of tools to the table, make sure and learn about existing web technologies, and put them to use across the API design process.
    - **Simplicity** - Consider simplicity at every turn when designing APIs, providing the small possible unit of value you possibly can--simplicity goes a long way.
    - **Consistency** - Employ consistent approaches to all aspects of API design, providing a familiar approach across all APIs published.
    - **Easy to Read** - While APIs are for computers, they should be easy to read by humans, making documentation more accessible.
    - **Easy to Learn** - Keeping APIs simple, and consistent, will contribute to them being easy to learn about for any potential API consumer.
    - **Hard to Misuse** - When APIs do one thing, and does it well, you reduce the opportunity for misuse, and people putting them to work in unintended ways.
    - **Audience Focused** - APIs should be designed with a specific audience in mind, providing a solution to a problem they are having.
    - **Experience Over Resource** - Make APIs reflect how they will be used, and experienced, over where the API came from and the resource it was derived from.
    - **Something You Will Use** - Always use your own APIs, allowing you to understand the challenges with integrating, as well as the pain of operations and outages.

- **Requests** - These are the core considerations when you are designing an API. They are extracted from publicly available, and some private API design guides of leading API platforms. They are not meant to be hard fast rules, but the core design elements to consider for organizations to consider along their own API journey.

    - **SSL** - Transport Layer Security (TLS) and its predecessor, Secure Sockets Layer (SSL), both of which are frequently referred to as 'SSL', are cryptographic protocols designed to provide

communications security over a computer network. Always support TLS / SSL by default when it comes to API operations, if at all possible.
    - **Host** - Put sufficient thought into what the baseURL will be for making API calls. Increasingly this element can also be dynamic, like other aspects of API design.
    - **Resource** - Consider how you craft your resources across all API operations. I do not subscribe to specific philosophies around this, just trying to provide a framework to think about this in.
    - **Verbs** - Learn about, and put all HTTP verbs to use when designing APIs.There is more than just GET, and even more than just POST, PUT, and DELETE.
    - **Parameters** - Standardize how parameters are crafted as part of API operations, using intuitive and consistent approaches.
    - **Headers** - Learn about, and use common headers for API requests. Study how the APIs you consumer put headers to work.
    - **Versioning** - Establish, and stick to a common versioning strategy, and use throughout API evolutions. Consider putting all versioning information within headers.
    - **Pagination** - Learn about common ways to paginate, and establish a single way to handle across API operations.
    - **Filtering** - Consider how filtering will happen across all APIs, and establish a single way to filter API resources.
    - **Sorting** - Establish a single approach to how API responses can be sorted, and use across all API resources.
    - **Granularity** - Always be aware of the granularity of your API endpoints, and each resource being exposed, keeping everything as small as possible.

- **Response** - What design considerations go into the response an API returns. It can be easy to focus on the request surface area for an API, but a response has a number of things to consider as well.

    - **Status Codes** - Learn about, and use HTTP status codes in a consistent way across all API operations.
    - **Error Handling** - Establish a single error handling strategy, and apply consistently across all API operations.
    - **Rate Limits** - Establish a single approach to rate limiting of API resources, and apply consistently across all API operations.
    - **UTF-8** - UTF-8 is a character encoding capable of encoding all possible characters, or code points, in Unicode. The encoding is variable-length and uses 8-bit code units. Make sure you use UTF-8 encoding for your API responses, supporting proper encoding.
    - **CORS** - Enable CORS for your API endpoints, providing the most flexibility possible in making API calls.
    - **JSONP** - Provide JSONP if you are unable to enable CORS, allowing for easier integrations.

- **Media Types** - A media type (also known as MIME type and content type) is a two-part identifier for file formats and format contents transmitted on the Internet, assigned by the Internet Assigned Numbers Authority (IANA). There are a handful of default media types that every API should consider as part of the API design process.

    - **text/html** - Provide HTML media types for API responses.

    - **application/json** - Provide JSON media types for API responses.

- **Open Standards** - There are a number of existing open standards that should be considered as part of the API design process. Designers and architects should resist reinventing the wheel when it comes to many of the aspects of API design, as things have already been done. These are just a handful of the open standards that are used by other companies in their API design.

    - **JSON Schema** - JSON Schema, describes your JSON data format. JSON Hyper-Schema, turns your JSON data into hypertext. Use JSON schema when possible to standardize and validate your JSON structure.
    - **iCalendar** - Use iCalendar when representing date / time formats in your API responses.
    - **vCard** - vCard is a file format standard for electronic business cards. vCards are often attached to e-mail messages, but can be exchanged in other ways, such as on the World Wide Web or instant messaging. They can contain name and address information, telephone numbers, email addresses, URLs, logos, photographs, and audio clips. Use vCard when representing contact data in your API responses.
    - **UUID** - Use a universally unique identifier (UUID) when possible.
    - **ISO 8601 (Date / Time)** - ISO 8601 Data elements and interchange formats – Information interchange – Representation of dates and times is an international standard covering the exchange of date and time-related data. The purpose of this standard is to provide an unambiguous and well-defined method of representing dates and times, so as to avoid misinterpretation of numeric representations of dates and times, particularly when data are transferred between countries with different conventions for writing numeric dates and times.
    - **ISO 4217 (Currency)** - Use ISO 427 data elements and interchange formats for the representation currency codes. ISO 4217 is a standard published by International Organization for Standardization, which delineates currency designators, country codes (alpha and numeric), and references to minor units in three tables: current currency & funds code list, current funds codes, list of codes for historic denominations of currencies & funds.
    - **ISO 3166 (Country)** - Use ISO 3166 data elements and interchange formats for the representation country codes. The purpose of ISO 3166 is to define internationally recognized codes of letters and/or numbers that we can use when we refer to countries and subdivisions. However, it does not define the names of countries .
    - **Schema.org** - Consider using Schema.org representations for common data elements.

- **Design Process** - Beyond request and response design concepts that go directly into the overall design of the API, there are many other elements that go into the overall design process to consider.

    - **Definitions** - Usage of common API definition formats like Swagger, API Blueprint, RAML, and others for describing and defining APIs.
    - **Editor** - An IDE for editing API definitions, as well as possible GUI interface for editing all of API details through desktop or web tooling.
    - **Forkable** - Allow an API definition to be forked, and built upon using a common API definition format.
    - **Sharing** - Allow for API definitions to be shared amongst team members through links, chat, email, and other channels.

    - **Collaboration** - Enable the collaboration between users, both technical, as well as business stakeholders.

- **Organization** - How is the API design process organized, centralized, or collaborated around? What services, tools, processes, and concepts are in play when it comes to the long term, as well as short term organization of the API design line along the life cycle.

    - **Guide** - Pull together a common API design guide for use across an organization and sharing with the public to demonstrate a standard approach is being used.
    - **Contact** - A common approach to defining and providing relevant contact information for each API resource crafted.

- **Other** - What are some of the other nickel and dime items, as part of design process that should be considered? This is my general catch-all bucket for the API design line, where I put links to other lines, stops, and just loose items that will affect API design, and should be thought about.

    - **Interactive Documentation / Console** - Generated API documentation allowing developers to make calls against APIs as they are learning about the interface, turning API education into a hands on experience.
    - **Github Sync** - The ability to store and sync API definitions with Github, providing a central public or private repository for the definition of an API resource.

Domain Name System (DNS)

DNS is the order in our API house. It is our lord, when it comes to Internet acronyms APIs are beholden to. The Domain Name System (DNS) is a hierarchical distributed naming system for computers, services, or any resource connected to the Internet or a private network. It associates various information with domain names assigned to each of the participating entities.

DNS is key to finding the digital resources we make available via APIs. There are a number of common DNS patterns emerging for operating APIs. There is also a number of API providers out there, who provide APIs for managing DNS, making the automation of API operations a reality. DNS is essential to a successful API operations, and having the right services and tools for managing API DNS is very critical aspect of any API operation.

- **Core DNS** - DNS is a central actor in the world of web APIs. While not immediately seen as active player of API life cycle, it is. Here are some of the core considerations when it comes to DNS.

    - **Domain** - Management of a single domain, and its administrative information.
    - **Record** - Management of all types of records for a single domain.
    - **Zone** - Manage the zones for any particular domain.
    - **Registration** - Being able to register new domains, purchase, and manage ownership and registrar.
    - **Cache** - Allow for domain level caching of all requests made through the domain.

    - **IP Address** - Ability to manage IP address information, and perform lookups.

- **Stability** - What does it take to achieve stability when it comes to DNS? What are the threats, and what tools and services are available to us, to make sure DNS is as stable as possible.

    - **Monitors** - The ability to monitor domain health and availability, and receive notifications, or take actions based upon.
    - **Threat Analysis** - Being able to establish and understanding of, and measure DNS level threats.

- **Utility** - What other utilities are there for managing DNS in support of APIs? These are some of the utility aspects of DNS operations.

    - **Statistics** - Providing statistics about DNS configuration for an operation through visual interface.
    - **Import** - Allowing for the import of DNS configuration using portable formats.
    - **Export** - Allowing for the export of DNS configuration using portable formats.

Containers

The latest virtualization evolution lead by container provider Docker, has had a significant impact on API deployment and operations, and again, inversely APIs have played an important role in container use. Like many other shifts in the tech landscape, APIs are a driving force of the explosion in the containerization of IT, and its overlap with other movements like Microservices, and Devops.

Like many other areas of this research containers can be a category of APIs, as well as a building block for API deployment and operations. While containers can be used for many other aspects of operations, this research focuses on how container APIs, can be used in the automation of various stops along a modern API life cycle. Looking at, and considering the API driven elements of container operations, tool and services available, and how APIs can be better provided, via small, modular, virtualized containers.

- **Core Elements** - Containers are making a big impact on API operations, and APIs play a big role in container adoption. What are some of the common considerations when it comes to containers and APIs.

    - **Containers** - A single unit of compute, complete with operating system, and application layer.
    - **Images** - A snapshot of a single container instance that then allows it to be deployed as a container.
    - **Nodes** - A single server instance, where many containers can be deployed and managed.
    - **Volumes** - A data volume is a specially-designated directory within one or more containers that bypasses the local.
    - **Clusters** - Allows for the creation and access to a pool of containers, organized into a meaningful group.
    - **Networks** - Defining the network linkage between containers.

    - **Hub** - A cloud-based registry service for building and shipping application or service containers.
    - **Registry** - A server location for registering, storing, and distributing container images.

- **API** - Container solutions all have APIs, which allow you to control all aspects of their deployment, operation, and deprecation.

    - **Containers** - A web API for managing containers.
    - **Image** - A web API for managing container images.
    - **Volumes** - A web API for managing the volumes used by containers.
    - **Networks** - A web API for managing container networks.

Virtualization

While virtualization plays a significant role in containers, this area of research focuses on how you virtualize API operations, and the resources made available via API technology. I wanted to look at how entire APIs are being replicated as sandbox environments, as well as how the resources themselves are be virtualized, allowing for dummy data, sample images, and templates of common resources.

While containers may be used as part of what virtualization occurs at tis API level, we'll be focusing on the API and resources themselves. As the Internet of Things expands, API virtualization is growing much more critical, providing sandbox, and templated environments that don't require real-world physical implementation, and more simulation. There are many approaches to providing sandboxes, simulations, and templates for the API lifecycle, and this is a snapshot of what is currently going on today.

- **Core Elements** - Containers are about managing virtualization at the compute level, where API virtualization is looking to focus on the virtualization of APIs themselves, and the data, content, and other resources that are being made available via APIs.

    - **Mock** - Creation of mock API interfaces that matches a specific API definition.
    - **Sandbox** - Generation of a sandbox environment that matches a specific API definition.
    - **Port Forwarding** - Forward of specific ports to a specific virtualized API instance.
    - **SSL** - Use of SSL when communicating with virtualized API instances.

- **Data Virtualization** - What are some of the considerations when it comes to data virtualization? How are providers, and platforms allowing for data to be virtualized, and made available via APIs.

    - **Templates** - Being able to create templates of data to be used when virtualized APIs are setup.
    - **Dummy Data** - Dummy data sets that can be imported individually or as collections into virtualized APIs.
    - **Excel Data** - Import of data from Microsoft Excel spreadsheets or Google sheets into virtualized APIs.

- **Import / Export** - Like every other area of the life cycle, API virtualization is being driven by common API definition formats. This is in duplicate of every other area of this research.

    - **Import Swagger** - Allow for the creation of virtualized APIs using existing Swagger definitions.
    - **Import RAML** - Allow for the creation of virtualized APIs using existing RAML definitions.
    - **Import API Blueprint** - Allow for the creation of virtualized APIs using existing API Blueprint definitions.
    - **Import WADL** - Allow for the creation of virtualized APIs using existing WADL definitions.
    - **Import Postman** - Allow for the creation of virtualized APIs using existing Postman definitions.

- **Other Elements** - Just a loose collection of the other virtualization things to consider as you push into this area.

    - **Reporting** - Providing reporting tools on virtualized instances, providing developers and consumers with snapshots of activity.
    - **Analytics** - Provide real-time, and other analytics that give details of what is happening with virtualized APIs as they operate.
    - **Teams** - Provide team tooling, allowing multiple individuals to work together when monitoring API virtualizations.

Deployment

Discussion and best practices around the actual deployment of APIs has been one of the more deficient areas of my API research. With growth in the space, we are beginning to see more discussion around the actual deployment of APIs, whether it is using gateway solutions, hand-crafted using open source frameworks, or the growing number of cloud service providers. API deployment comes in many shapes and sizes, and means very different things to many different developers, architects, IT operators, and increasingly business leaders who are making API driven business decisions.

This research began looking at what open source frameworks were available, but then I noticed that API management, and API design service providers were beginning to offer assist in the deployment process using common gateways. Along the way I have tracked on multiple waves of cloud service providers who are looking to assist in the deployment of APIs from common data sources, including database, spreadsheets, or XML data files. More recently I'm seeing a new breed of "serverless" providers, enabling quick, and easy deployment of modular API resources using cloud virtualization technologies. This research is dedicated to keeping a track on this shifting and rapidly expanding area of the API lifecycle.

- **General Considerations** - What are the considerations for API developers, specifically when it comes to the deployment. This portion of the research is meant to focus on how we actually launch an API out of the design and definition, into the API management space.

    - **CSV to API** - Deploying an API from comma separated files, providing a simple, delimited, structured data store as a file rather than database.

    - **Database to API** - Connecting to a database and generating a CRUD, or create, read, update and delete API on an existing data source.
    - **Framework** - Crafting and API using an existing API framework in a particular programming language, providing the HTTP scaffolding needed.
    - **Gateway** - API gateways provide a solution for tapping into internal systems and connecting with external platforms, to expose APIs.
    - **Proxy** - API proxy are common place for taking an existing API interface, running it through an intermediary which allows for translations, transformations and other added services on top of API.
    - **Connector** - Contrary to an API proxy, there are API solutions that are "proxyless", while just allowing an API to connect or plugin to the advanced API resources
    - **Hosting** - Hosting is all about where you are going to park your API. Usual deployments are on-premise within your company or data center, in a public cloud like Amazon Web Services or a hybrid of the two.
    - **Scraping** - Harvesting or scraping of data from an existing website, content or data source, when an existing data source is unavailable, and then publishing as an API.
    - **Container** - The new virtualization movement, lead by Docker is providing new ways to package up APIs, and deploy as small, modular, virtualized containers.
    - **Github** - Provides a simple way to publish documentation, code libraries, TOS, and other common building blocks in support of API operations.
    - **JSON to API** - Deploying an API from JSON file, providing a simple, delimited, structured data store as a file rather than database.

Management

This is the oldest area of my API Evangelist area, and is the area that has spun out every other research project in this paper. It all started studying how popular API providers like Twitter, and Google were managing their operations, pulling the common building blocks from their publicly available developer portals. It then evolved into tracking the features of services and tooling being sold by API service providers, targeting the growing number of API providers--the result is a long list of essential elements of API management in 2016.

There are many types of APIs, so there is no one-size fits all strategy for any of the areas covered by this research, it is intended to be a buffet of ideas, that API providers might find valuable to their operations. This section has the essential elements that apply to not just a single API, but potentially across all APIs, or individual groups of APIs. Some of the management building blocks may be one-time considerations, with other having a more recurring, and on-going function in API management workflows--this is just an exploration of what the possibilities are.

- **Onboarding** - What is the process for onboarding of new users? Walk through what a new user will experience, looking at each step from landing on home page, to having what I need to make my own API call. Reduce as much friction as I can, and making on-boarding as fast as possible.

    - **Portal** - A simple, easily accessible portal for engaging with API consumers, providing a single point of entry for all API engagements.

    - **Getting Started** - Laying out simple, clear steps for developers on how to register, authenticate, access documentation and code samples, and get support and any other details that are essential to integration.
    - **Self-Service Registration** - Providing a self-service registration for a developer portal, allowing developers to signup and get access to an API, 24 hours a day, 7 days a week.
    - **FAQ** - Providing developers a list of the common questions asked of an API platform, with simple answers to each question, allowing them to understand the common problems faced during onboarding.
    - **Sign Up Email** - Providing a simple, informative, personal email immediately upon signup, letting developers know what they need to onboard.

- **Documentation** - What is provided when it comes to documentation for the platform? There are a number of proven building blocks available when it comes to API documentation.

    - **Documentation** - Simple, easy to understand, documentation of what an API does, that is kept up to date with API roadmap.
    - **Interactive Documentation** - Interactive, hands-on documentation, provided by common approaches like Swagger and API Blueprint.
    - **Error Response Codes** - Provide a robust, complete, clear and meaningful list of API error response codes.

- **Authentication** - Authentication is central to many other lines of the API lifecycle. There are several common elements present in modern API solutions that address authentication.

    - **Authentication Overview** - An overview of how authentication is handled for a platform, and which formats are supported, setting expectations for consumers.
    - **Key Access** - Providing simple tokens, often called API key as a common way to access to APIs, issuing one to each developer and per application they register.
    - **Basic Auth** - Usage of the basic authentication format that is part of the standard HTTP operations, employing a username and password as credentials for accessing API resources.

- **Code Management** - What resources are available for managing code across the platform. This area focuses on just the services, tooling, and process associated with code management, not always the code itself.

    - **Github** - Using Github for managing of code samples, libraries, SDKs, and other supporting elements of an API platform.
    - **Code Page** - A page dedicated to providing access to code resources, whether samples, libraries, SDKs, PDKs, or starter projects.
    - **Community Supported Libraries** - In addition to providing your own company developed libraries, showcasing the libraries of trusted developers fro within the API community.
    - **SDKs.io** - Making sure your profile on SDKs.io is complete, with up to date SDKs available so users can use in their integration.

- **Self-Service Support** - What support services are available 24/7, that developers can take advantage of without requiring the direct assistance of platform operators.

    - **Stack Overflow** - Using the public Q&A site Stack Overflow as part of support operations for an API.
    - **Knowledgebase** - Providing a single repository of content, organized by title, and tag, allowing you to search for keywords and phrases, as well as browse for answers.

- **Direct Support** - What support services are available that developers can take advantage of, that involves direct employee attention.

    - **Email** - Providing a simple email address that API consumers can use when looking to get answers to their questions.
    - **Contact Form** - A contact form available in the central API portal, that developers can use to ask questions.
    - **Calendar** - A single calendar of all events related to API operations, providing a place where developers can find all related goings on.

- **Communications** - What are the communication elements available as part of the overall feedback loop for an API platform. There should be at least a minimum viable communications present, otherwise it is unlikely anyone will learn that a platform exists.
    - **Blog** - Providing a blog dedicated to API platform operations, with stories about the providers, consumers, and other related stories.
    - **Blog RSS Feed** - Make sure there is an RSS feed for any blog that is published as part of API operations.
    - **Twitter** - Provide an active Twitter account for support, as well as evangelism efforts around an APIs operations.
    - **LinkedIn** - Provide an active LinkedIn account for support, as well as evangelism efforts around an APIs operations.
    - **Email** - Provide an email for not just support, but also general platform communications.

- **Updates** - How are we planning, and communicating updates to the platform? Providing a map of how things have changed across the platform, from versioning of the API itself, to even documentation, and other aspects of platform operations.

    - **Status Dashboard** - A single location, at dedicated subdomain, providing status updates for all API resources, giving developers a place to go when looking to see if platform is available.
    - **Roadmap** - A detailed list of what additions and changes are being planned as part of API operations, giving consumers a place where they can go learn about what is next in an APIs evolution.
    - **Change Log** - A detailed list of changes have been made to an API platform, giving consumers a single place they can go to learn about what changes have been made.
    - **Status RSS** - Providing an RSS feed for a status dashboard, allowing users to receive updates on any website, and via any RSS client.

- **Resources** - What other resources are available for API consumers to take advantage of? Common resources provide a wealth of usually self-service knowledge resources that API consumers can consume on demand, as part of their API integration journey.

    - **Case Studies** - Offering a wealth of case studies of actual implementations that have occurred with an API , showing potential consumers what is possible.
    - **How-to Guides** - Providing how-to guides that walk consumers through common aspects of API integration, giving them self-service resources they can use.
    - **Events** - Publish a page with information about any events the an API program will be attending, or maybe links to video and slide decks from past conferences.

- **Research & Development** - APIs are often R&D departments for companies, organizations, institutions, and even within the government. What are some of the ways that these organizations are pushing forward R&D using APIs? There are a number of common elements to consider.

    - **Labs** - Offering a labs environment, that showcases ideas that are being developed internally and by partners, possibly even providing opportunities for public consumers.
    - **Ideas** - Like showcase existing applications, an idea showcase is about providing a place where new and existing ideas can be shared, and potentially built upon by the community.

- **Environment** - What environments are available for working with API resources? This area overlaps with existing containerization, and virtualization research. How are we providing development vs production environments, and how are we applying environments to not just APIs, but also data content.

    - **Sandbox** - Provide developers a sandbox environment to develop and test their code, reducing headaches for consumers when developing their applications.

- **Developer Account** - Consumers of an API platform always need an account where they can get access to API authentication, usage reports, and other common elements of API operations. What does the developer account, or area look like, and what resources are available for developers to take advantage of.

    - **Developer Dashboard** - Provide developers a single dashboard for getting at all their tools, metrics and information they need to successfully manage their usage.
    - **Account Settings** - Provide developers access to their basic account detail and settings, giving them quick access to their account configuration and allow for easy updates.
    - **Reset Password** - Provide the necessary tools for developers to gain access to their account if they lose their password.
    - **Application Manager** - Providing an interface that allows developers to manage one or many applications that will be integrating with an API.

    - **Usage Logs & Analytics** - Allowing developers to be able to see where they stand with usage, logging and other aspects of operations, by providing them reports and analytics of their history.
    - **Billing History** - Provide clear and easy access to what a developer has been billed, allowing them to access and download their billing history around API usage.
    - **Message Center** - Providing developers with a messaging system within their developer accounts, and communicate with API providers, and receive system updates.
    - **Delete Account** - Giving developers the ability to delete their account from within their account portal.
    - **Service Tier Management** - Allow developers to change the service tier they operate with in via their developer dashboard, without having to request changes from platform support.

- **Reciprocity** - What interoperability, automation, and reciprocity opportunities are available via the API platform? There are a number of technical, business, and political elements that make interoperability and automation possible in a way that enable reciprocity to exist between platform operators, developers, and end-users.

    - **Data Portability** - Providing users with the ability to get data out of a system through a bulk download and via an API in a usable format that employs open standards.
    - **Automation** - Offer ready to go integrations with popular API automation platforms like Zapier and IFTTT, allowing any user to migrate data between platforms.
    - **Integrations** - Integrations showcase other 3rd party platforms that an API provider is already connected to, providing ready to go platform integration solutions that any developer can take advantage of.

- **Corporate** - What are the corporate considerations for platform operations? Showcasing the team, and individual faces behind API operations at an organization. There are a handful of common approaches to providing the corporate face of API operations.

    - **Mission** - What is the mission of your company, organization, or government agency behind the platform.
    - **Team Showcase** - Provide a simple team page, showcasing the team behind the API, including photos, personal stories or bios, and any relevant social media accounts like Twitter and Github.

Monitoring

Are APIs available? Can API consumers depend on them? These are the questions being answered using modern approaches to API monitoring. This is an attempt in the aggregation of the common practices, services, and tools in one place, so that they can be applied anywhere. The API monitoring building blocks I am making available, may be used to simply notify API providers of outages, all the way to publicly sharing history, and using data to drive industry level rating systems.

This API monitoring research overlaps with other areas of API testing, and performance, which is not included in this essential report. The objective of this research is to ensure a minimum viable API

monitoring is present for all API operations, with an emphasis on expanding upon, as resources become available. There are a new wave of API monitoring service providers, that are making API monitoring much more affordable, and efficient using more automation, which like other areas of this research, is increasingly done using APIs.

- **Core Details** - What are the core elements to API monitoring that I am finding used by API providers, and offered by API service providers.

    - **Request Editor** - Being able to edit the request being made as part of API monitoring process.
    - **Request Retry** - Being able to retry a specific request that has been made at previous timeframe.
    - **Request Sharing** - Allow the sharing of a specific request designated for monitoring.
    - **Request Playback** - Enable the playback of a specific request that has been made at previous timeframe.
    - **Request Scheduling** - Allow the scheduling of when to run specific task at specific time(s).
    - **Request Compare** - Being able to compare to requests against each other.
    - **Request Automation** - The ability to automate the running of multiple requests.
    - **Service Availability** - Making sure a service is available.
    - **Latency Measurement** - Measuring the latency associated with a specific request.
    - **Response Header Validation** - Being able to validate for a specific header for a request.
    - **Response Body Validation** - Being able to validate the content of the body for a request.

- **Management Monitoring** - Monitoring is not always about monitoring the APIs themselves. This section explores the monitoring of other aspects of API operations.

    - **Documentation Monitoring** - Keeping track of changes to an APIs documentation
    - **Pricing Monitoring** - Notifications when an API platform's pricing changes
    - **Terms of Service Monitoring** - Updates when a company changes the terms of service

- **Targeted Monitoring** - How can we monitor from various dimensions like location or with specific providers.

    - **Provider Based Monitoring** - Being able to monitor an LAPI from a particular provider.
    - **Region Based Monitoring** - Begin able to monitor from a specific geographical region.
    - **Public Monitoring** - Monitoring of public APIs, and providing the information for others to use.

- **Authentication** - What authentication approaches are available to us? Do our API monitor tools and services cover all the authentication schemes we will use across platforms.

    - **Basic Auth** - Using Basic Auth, a username and password for authentication.
    - **OAuth** - Using OAuth for platform authentication.
    - **API Keys** - Using API keys for authentication.

- **Utility** - What are the other utilities available to use as part of the monitoring process.

    - **Collections** - Being able to organize requests into collections.
    - **Localhost** - Allow for executing monitoring requests using localhost.
    - **Teams** - Enable the ability for teams to work together and share monitoring.
    - **API** - Providing an API for API monitoring tools.

- **Notification** - Monitoring is all about awareness, so notifications is an important part. What are some of the common ways API monitoring notifies either provider or consumers of an API.

    - **SMS** - Allow notifications to be received by SMS.
    - **Email** - Allow notifications to be received by email.
    - **Webhook** - Allow notifications to be received by webhook.

- **Import** - What options are available for importing, and exporting of API monitoring test, details or other aspects of API monitoring. There are some common formats emerging, that are used for allowing the import and export API monitors.

    - **Postman** - Allow for importing using Postman Collections.
    - **Swagger** - Allow for importing using Swagger.

- **Reporting** - Reporting is an essential step when it comes to monitoring, providing real-time to historical access to API monitoring information and visualizations.

    - **Dashboard** - Providing a dashboard to review monitors.
    - **Analytics** - Providing analytics around API monitoring.

- **3rd Party** - APIs are all about 3rd party integration, and it is common for API monitoring services to provide ready to go integrations, to integrate services with other platforms you depend on. These are some of the common 3rd party integration elements present in API monitoring services.

    - **Slack** - Provide integration with Slack.
    - **PagerDuty** - Provide integration with PagerDuty.
    - **VictorOps** - Provide integration with VictorOps.
    - **HipChat** - Provide integration with HipChat.
    - **Flowdock** - Provide integration with Flowdock
    - **OpsGenie** - Provide integration with OpsGenie.

Security

As the number of APIs grows, and the importance of data and resources made available them becomes more mission critical, the need for API security will exponentially grow. APIs alone, bring their own dimension to security. Just knowing where resources are, and that they are available a single HTTP

endpoint, allows for the auditing, and securing of digital resources. This is the first dimension of security, that APIs afford, but since they also utilize HTTP as the transport, there are a number of other considerations.

Borrowing from website, web and mobile application security practices, one can easily build a pretty robust list of considerations for securing APIs. This research looks to map out what the most common security concerns are, and provide a structured way to approach overall API operations security, as well as the individual security of specific API driven resources. API specific security resources aren't as available in the quantities that they should be in such a volatile environment, and this research is looking to change that--hopefully pushing forward the number of available services, and supporting information.

- **Auth Formats** - When it comes to security testing, we'll need access to the full spectrum of authentication formats, so that all aspects of integration can be probed and tested.

    - **Basic Auth** - Using Basic Authentication for security access to API resources.
    - **OAuth** - Using OAuth for authorization and access management for API resources.
    - **API Keys** - Using API keys for securing access to API resources.
    - **JSON Web Token** - Using JSON web tokens for securing access to API resources.

- **Auth Considerations** - Beyond the formats themselves, what are some of the considerations when it comes to security and authentication?

    - **Session Management** - API should pass session-based authentication, either by session token via a POST or by API key as a POST body argument or as a cookie, avoiding usernames, passwords, session tokens, and API keys in the URL
    - **Session State** - Many web services are written to be as stateless as possible, usually ending up with a state blob being sent as part of the transaction.
    - **Methods Whitelist** - Properly restrict the allowable verbs such that only the allowed verbs would work, while all others would return a proper response code (for example, a 403 Forbidden).
    - **Cross-Site Request Forgery** - Make sure any PUT, POST, and DELETE request is protected from Cross Site Request Forgery, typically done using a token-based approach.
    - **Insecure Direct Object References** - It may seem obvious, but if you had a bank account REST web service, you'd have to make sure there is adequate checking of primary and foreign keys.

- **Input Validation** - Focusing specifically on the area of input, and making sure only safe data and content is being able to be input. What are the specific input security considerations?

    - **Assist the User** - Assisting in the input of high quality data into APIs, by validating what is submitted, and rejecting anything that does not pass validation.
    - **Validate Content-Types** - The server should never assume the Content-Type; it should always check that the Content-Type header and the content are the same type.

    - **Validate Response Types** - Do NOT simply copy the Accept header to the Content-type header of the response, and reject the request if the Accept header does not specifically contain one of the allowable types.
    - **JSON Validation** - Making sure all JSON is valid helps make sure that APIs are operating as expectation and making sure vulnerabilities are not being passed in with API requests and responses.
    - **XML Validation** - XML-based services must ensure that they are protected against common XML based attacks by using secure XML-parsing.

- **Output Validation** - Like the input, is the return output what it should be? Let's go through some of the considerations when it comes to output validation.

    - **Send Security Headers** - The server should always send the Content-Type header with the correct Content-Type, and preferably the Content-Type header should include a charset.
    - **JSON Encoding** - Use a proper JSON serializer to encode user-supplied data properly to prevent the execution of user-supplied input on the browser.
    - **XML Encoding** - Do not assemble XML string concatenation, rather us a XML serializer, ensure that the XML content sent to the browser is parseable and does not contain XML injection.
    - **Link Integrity** - Checks the reputation of web links in real time, providing an invisibly secure experience by blocking malicious and unwanted links from being present or loading in content.

- **Transport Level Security** - Are we considering everything beyond the request and the response? Are we allowing for the proper security in transport?

    - **Data in Transit** - Unless the public information is completely read-only, the use of TLS should be mandated, particularly where credentials, updates, deletions, and any value transactions are performed.

- **Abuse of Functionality** - Looking to where intruders are abusing the functionality of the API, to get the results they are looking for.

    - **Buffer Overflow Attack** - Avoid overwriting of memory fragments of the process, values of the IP (Instruction Pointer), BP (Base Pointer) and other registers which can cause exceptions, segmentation faults, and other errors to occur.
    - **Buffer Overflow via Environment Variables** - Avoid this pattern that involves causes a buffer overflow through manipulation of environment variables, which once the attacker finds that they can modify an environment variable, they may try to overflow associated buffers.
    - **Overflow Binary Resource File** - Coming from input data, and an Overflow Binary Resource File, where attacker modifies/prepares the binary file in such a way that the application, after reading this file, becomes prone to a classic Buffer overflow attack.

- **Data Structure Attacks** - How are we looking for an attacker manipulating and exploiting characteristics of system data structures in order to violate the intended usage and protections of

these structures.

    - **Cross-Site Request Forgery (CSRF)** - An attack that forces an end user to execute unwanted actions on a web application in which they're currently authenticated, targeting state-changing requests, not theft of data, since the attacker has no way to see the response to the forged request.
    - **Logic/time Bomb** - A logic bomb is a piece of malicious code that executes when specific trigger conditions are met, such as a date or time, or possible a specific database event.
    - **Trojan Horse** - A Trojan Horse is a program that uses malicious code masqueraded as a trusted application, which is vehicle in which malicious code can be injected on benign applications, masqueraded in e-mail links, or JavaScript.
    - **Cross-Site Request Forgery (CSRF)** - An attack that forces an end user to execute unwanted actions on a web application in which they're currently authenticated, targeting state-changing requests, not theft of data, since the attacker has no way to see the response to the forged request.
    - **Execution After Redirect (EAR)** - Execution After Redirect (EAR) is an attack where an attacker ignores redirects and retrieves sensitive content intended for authenticated users. A successful EAR exploit can lead to complete compromise of the application. - https://www.owasp.org/index.php/Execution_After_Redirect_(EAR)
    - **Session Fixation** - An attack that permits an attacker to hijack a valid user session, and explore a limitation in the way the web application manages the session ID, more specifically the vulnerable web application.
    - **Session Hijacking Attack** - An attack that consists of the exploitation of the web session control mechanism, which is normally managed for a session token, exploiting http communication which uses many different TCP connections.
    - **Session Prediction** - An attack that focuses on predicting session ID values that permit an attacker to bypass the authentication schema of an application, by analyzing and understanding the session ID generation process, an attacker can predict a valid session ID value and get access to the application.

- **Embedded Malicious Code** - A developer might insert malicious code with the intent to subvert the security of an application or its host system at some time in the future. It generally refers to a program that performs a useful service but exploits rights of the program's user in a way the user does not intend.

    - **Parameter Delimiter** - An attack based on the manipulation of parameter delimiters used by web application input vectors in order to cause unexpected behaviors like access control and authorization bypass and information disclosure.
    - **Resource Injection** - This attack consists of changing resource identifiers used by an application in order to perform a malicious task, allowing data to be manipulated to execute or access different resources.
    - **Server-Side Includes (SSI) Injection** - An attack that allows the exploitation of a web application by injecting scripts in HTML pages or executing arbitrary codes remotely, exploiting through the manipulation of SSI in use and force its use through user input fields.

    - **SQL Injection** - The insertion of a SQL query via the input data allowing the reading sensitive data from the database, modify database data, execute administrative tasks, and in some cases issue commands to the operating system.
    - **Web Parameter Tampering** - An attacked based on the manipulation of parameters exchanged between client and server in order to modify application data, such as user credentials and permissions, price and quantity of products, etc.
    - **XPATH Injection** - An attack occurring when a web site uses user-supplied information to construct an XPath query for XML data, sending malformed information to an API opening up how the XML data is structured, or access data.
    - **Code Injection** - Attack types which consist of injecting code that is then interpreted/executed by the application, exploiting poor handling of untrusted data, and usually made possible due to a lack of proper input/output data validation.
    - **Command Injection** - An attack in which executes arbitrary commands to the host operating system via a vulnerable application, made possible when an application passes unsafe user supplied data (forms, cookies, HTTP headers etc.).
    - **Content Spoofing** - An attack made possible by an injection vulnerability, where an API does not properly handle user supplied data, and an attacker can supply content, typically via a parameter value, that is reflected back to the user.
    - **CORS RequestPreflighScrutiny** - Opening up the possibility to expose resources to all or restricted domain, made by AJAX request for resource on other domain than is source domain.
    - **Cross-site Scripting (XSS)** - A type of injection, in which malicious scripts are injected into otherwise benign and trusted web sites, occurring when an attacker send malicious code, generally in the form of a browser side script, to a different end user.
    - **Format String Attack** - Occurring when the submitted data of an input string is evaluated as a command, allowing the execution of code, to read the stack, or cause a segmentation fault, causing behaviors that could compromise the system.
    - **Full Path Disclosure** - Vulnerabilities enabling the attacker to see the path to the webroot/file, potentially opening up full access to the underlying system.

- **By Force** - What can be injected to adversely affect platform operations?

    - **Brute Force Attack** - A brute force attack can manifest itself in many different ways, but primarily consists in an attacker configuring predetermined values, making requests to a server using those values, and then analyzing the response.
    - **Cash Overflow** - An attack specifically aimed at exceeding hosting costs, either essentially bankrupting the service owner or exceeding the cost limits, leading the cloud service provider to disable the application.
    - **Denial of Service** - The Denial of Service (DoS) attack is focused on making a resource (site, application, server) unavailable for the purpose it was designed.

- **Path Traversal Attack** - A path traversal attack (also known as directory traversal) aims to access files and directories that are stored outside the webroot folder. What is being done to address this are.

    - **HTTP Request Smuggling** - The HTTP Request Smuggling attack explores an incomplete parsing of the submitted data done by an intermediary HTTP system working as a proxy.

    - **HTTP Response Splitting** - Occurs when data enters a through an untrusted source, most frequently an HTTP request, included in an HTTP response header sent to a web user without being validated for malicious characters.
    - **Traffic Flood** - A type of DoS attack that explores the way that the TCP connection is managed, with the generation of a lot of well-crafted TCP requisitions, with the objective to stop the Web Server or cause a performance decrease.

- **Probabilistic Technique** - An attacker utilizes probabilistic techniques to explore and overcome security properties of the target that are based on an assumption of strength due to the extremely low mathematical probability that an attacker would be able to identify and exploit the very rare specific conditions under which those security properties do not hold.

    - **Asymmetric Resource Consumption** - Asymmetric resource consumption consists in an attacker forcing a web application to consume excessive resources when the application fails to release, or incorrectly releases, a system resource.
    - **Cash Overflow** - An attack specifically aimed at exceeding hosting costs, either essentially bankrupting the service owner or exceeding the cost limits, leading the cloud service provider to disable the application.
    - **Denial of Service** - The Denial of Service (DoS) attack is focused on making a resource (site, application, server) unavailable for the purpose it was designed.

- **Protocol Manipulation** - An adversary takes advantage of weaknesses in the protocol by which a client and server are communicating to perform unexpected actions.

    - **Comment Injection Attack** - Comments injected into an application through input can be used to compromise a system, where data is parsed, an injected/malformed comment may cause the process to take unexpected actions that result in an attack.
    - **Double Encoding** - Encoding user request parameters twice in hexadecimal format in order to bypass security controls or cause unexpected behavior from the application.
    - **Forced Browsing** - An attack that enumerates and opens up access to resources that are not referenced by the API, allowing the discovery unlinked contents in the domain directory, such as temp directories and files, and configuration files.
    - **Path Traversal** - A Path Traversal attack aims to access files and directories that are stored outside the webroot folder, by browsing the application, the attacker looks for absolute links to files stored on the web server.
    - **Relative Path Traversal** - This attack is a variant of Path Traversal and can be exploited when the application accepts the use of relative traversal sequences such as "../".
    - **Repudiation Attack** - A repudiation attack happens when an application or system does not adopt controls to properly track and log users' actions, thus permitting malicious manipulation or forging the identification of new actions.
    - **Setting Manipulation** - The modification of settings in order to cause misleading data or advantages on the attacker's behalf, manipulating values in the system and manage specific user resources of the application or affect its functionalities.
    - **Unicode Encoding** - Opening up of flaws in the decoding mechanism implemented by decoding Unicode data format, allowing for encoding of certain characters in the URL to

bypass filters, thus accessing restricted resources.

- **Resource Depletion** - An attacker depletes a resource to the point that the target's functionality is affected. What are some of the common considerations for resource depletion?

    - **Cash Overflow** - An attack specifically aimed at exceeding hosting costs, either essentially bankrupting the service owner or exceeding the cost limits, leading the cloud service provider to disable the application.
    - **Cross-Site Request Forgery (CSRF)** - An attack that forces an end user to execute unwanted actions on a web application in which they're currently authenticated, targeting state-changing requests, not theft of data, since the attacker has no way to see the response to the forged request.
    - **Man-in-the-Middle Attack** - The interception of communication between two systems, where once the TCP connection is intercepted, the attacker acts as a proxy, being able to read, insert and modify the data in the intercepted communication.

- **Other Security Considerations** - Other elements that can be used as part of overall API security planning and execution. There are a number of things that I'm seeing out there, but it doesn't have its own category to exist in yet.

    - **Certification** - Providing a system for validating and certifying an APIs security, scanning the surface area and providing guarantees that the endpoint(s) are secure.
    - **Security Visualization** - Allowing API owners with reports and charting, allowing them to visualize the surface area and security of an API. Providing a single way to see all the potential attacks, vulnerabilities, and what is being done about security overall.
    - **Endpoint Tagging** - Allow for the tagging and organizing of APIs into specific groups for scanning, monitoring, and reviewing of APIs. Giving providers an easy way to organize so that they can be secured, and understood.
    - **Intrusion Correlation** - Providing the ability to connect the dots between various security attacks and intrusions, providing insight into patterns used by attackers, locations of attacks, and other details that might help security.
    - **Risk Scoring** - Offer a single scoring approach to be able to score attacks, based upon existing understanding of the space, but also historical data, providing a clear way to rank, and understand how vulnerabilities, and attacks could affect operations.
    - **Publish Your Page** - Publishing a single page that covers what the security approaches are for a platform, providing as much detail as possible about what practices are employed, what the schedule looks like, and what results of security history is.

Terms of Service

The legal terms of service are driving everything on the Internet today. They are essential in setting the tone of any API platform, setting the best practices for how API consumers engage with API resources, while also protecting the best interests of platform providers. There are a number of common patterns being established in how terms of service for APIs are crafted, published, and wielded as part of API operations.

This research does not intend on being a legal resource for API terms of service. This research intends to aggregate a number of the common aspects of API terms of service, in a plain english way. Like the technical layers can hide potential complexity of platform operations, the legal layers can hide complexity as well. We need more light shed on the legal side of API operations, and this checklist of common terms of service building blocks, is intended to begin shining a light on the terms of service driven portion of API operations.

- **Core Considerations** - I only have a single bucket for my API terms of service research. I'm still gathering data about terms of service for popular providers, and eventually, I will have more of a breakdown by category.

    - **Sites Covered** - What sites are covered via the terms of use. Even if there is a single, provide a basic reference to the domain. If there are multiple domains, make sure and list them. Also consider subdomains that might be a consideration.
    - **Children's Privacy** - Are there any special considerations for children's privacy. Will minors potentially be using the system, and what areas should be considered?
    - **Links to Non-Operators Web Sites** - What are the levels of responsibility and liability when it comes to external links made available through platform operations.
    - **Non-Personal Information** - What are the considerations for all other content and data areas. This not personally identifiable information, but everything else.
    - **Log Files** - Providing some details on how log files are used in platform operations, what is logged, and what isn't, and how things are collected, retained and shared.
    - **Cookies** -What are the cookie policies? How are cookies crafted, stored, and deprecated. Cover the cookie policies for a platform, and any connection to the API.
    - **Web Beacons** - Are web beacons in use? Are they return as part of API operations, and possibly required in all web and mobile deployments
    - **Personal Information** - What personal information is collected, stored, shared, and put to use? What are the security procedures, and other considerations for dealing with privacy of users personal information.
    - **Partner Resources** - Are there any members-only and partner specific opportunities, and how do these different levels of access impact terms of service.
    - **How We Use Your Information** - Be clear about how information is used. Is it only used as part of operations. Provide straightforward explanations of how information is put to work to power the platform.
    - **Information Sharing** - How is information shared with partners, government, and any potentially external partners. Does this occur via the API, and how does this affect terms of service.
    - **Access To Information** - What levels of access are there. Is everything available via APIs or downloads, or are there restrictions around what you can get access to.
    - **Accuracy of Information** - What are the expectations for accuracy of information available via the platform. What is the bar, and how is this level determined and enforced.
    - **Security** - What level of security is guaranteed as part of platform operations? How does the overall security practices impact the terms of service between platform and consumers?
    - **Opting Out** - What are the possibilities for opting out of services? Are there data sharing, licensing, or communication areas that can be opted out, or NOT opted out of.

Privacy

When done right APIs provide transparency into the pipes that are increasingly driving the web, mobile, and device applications we are increasingly depending on. This transparency, when coupled with healthy security practices, can provide platform providers, API consumers, and end-users of the applications and integrations built upon them, a better look at whether platform practices towards privacy are healthy or not.

Similar to terms of service, this research does not intend on being a legal resource for API privacy issues and policies. The goal with this research is to list out the common concerns when it comes to privacy, for all three legs of operations, the provider, consumer, and end-users. APIs can just as easily be used to damage privacy, as they can to assist in helping address privacy concerns. It is important to understand how privacy is being addressed by providers, as well as what the wider concerns for everyone else involved are.

- **Things That Impact Privacy** - Privacy is a new area that I have spun out of my research, and only have a general grouping considerations, as I till try and learn how privacy is being impacted as part of API operations.

    - **License** - What licensing considerations are there that could potentially impact the privacy of platform owners, consumers, and end-users.
    - **Intellectual Property Rights** - Where does intellectual property rights come into play, that could potentially impact the privacy of platform owners, consumers, and end-users.
    - **Permitted and Prohibited Uses** - What are the permitted uses of data and content made available via the APIs, and what are some of the common prohibited uses that impact the
    - **Use of Personally Identifiable Information** - Linking to the same area in terms of service, how is privacy considered when personally identifiable information is put to use. Lay out what is considered in this area.
    - **User Submissions** - What is the general view of user submissions, and user generated content and data. Is it core to platform operations, and what is the general tone of users and API consumers access and ownership of this exhaust.
    - **Technical Requirements and Limitations** - What are the technical requirements and limitations that impact the privacy of platform owners, consumers, and end-users.
    - **User Discussion Lists and Forums** - What privacy considerations have been made around discussion lists and forums operated by the platform? What data is stored? How is the privacy of users addressed?
    - **Liability** - What is the liability assumed by the platform, or possibly consumers, and end-users when it comes to platform operations, in the area of privacy.
    - **Termination** - What privacy considerations are taken when accounts are terminated. Can users, and API consumers terminate their relationship and assume that privacy is respected.
    - **Changes** - How may changes to the platform, APIs, and how data is collected and stored that may impact privacy.

Licensing

The legal side of API operations has a presence in almost all layers of API integration, from the server code, to client code that is used within applications and external systems. There are an ever growing number of licensing concerns, and approaches to applying, and enforcing licensing, and this is a look at that landscape, in an attempt to better understand.

I bundle many different types of licensing under one umbrella, from code licenses, to copyright licensing. Due to recent evolution in API related licensing cases like the Oracle v Google API copyright case, the landscape has only gotten more confusing. With the very technical nature of of APIs, many feel that code licensing covers all aspects, but when you look closer, the space is expanding, and copyright, software licenses, and even patents are coming into play, and need consideration too.

- **Server Code** - For many APIs, your server code will be your secret sauce and kept proprietary, but for those of you who wish to open source this critical layer, here are some options. To help you navigate the licensing, I recommend using Github's Choose a License.

    - **Apache** - The Apache License is a free software license written by the Apache Software Foundation (ASF). The Apache License requires preservation of the copyright notice and disclaimer. Like other free software licenses, the license allows the user of the software the freedom to use the software for any purpose, to distribute it, to modify it, and to distribute modified versions of the software, under the terms of the license, without concern for royalties. - http://www.apache.org/licenses/LICENSE-2.0
    - **GPL** - The GNU General Public License (GNU GPL or GPL) is the most widely used[6] free software license, which guarantees end users (individuals, organizations, companies) the freedoms to run, study, share (copy), and modify the software. Software that allows these rights is called free software and, if the software is copylefted, requires those rights to be retained. - http://www.gnu.org/licenses/gpl-3.0.en.html
    - **MIT** - The MIT License is a free software license originating at the Massachusetts Institute of Technology (MIT). It is a permissive free software license, meaning that it permits reuse within proprietary software provided all copies of the licensed software include a copy of the MIT License terms and the copyright notice.

- **Data** - Serving up data is one of the most common reasons for deploying an API, and the Open Data Commons, provides us with some licensing options.

    - **Public Domain Dedication and License (PDDL)** - The PDDL places the data(base) in the public domain (waiving all rights).
    - **Attribution License (ODC-By)** - You are free to share, create, and adapt, as long as you attribute the data source.
    - **Open Database License (ODC-ODbL)** - You are free to share, create, and adapt, as long as you attribute the data source, share-aloe, and keep open.

- **Content** - Separate from the data, APIs are being used to server up short, and long form content, where liberal Creative Common licenses should be considered.

    - **Attribution (CC BY)** - This license lets others distribute, remix, tweak, and build upon your work, even commercially, as long as they credit you for the original creation. This is the most accommodating of licenses offered.
    - **Attribution-ShareAlike (CC BY-SA)** - This license lets others remix, tweak, and build upon your work even for commercial purposes, as long as they credit you and license their new creations under the identical terms. This license is often compared to copyleft free and open source software licenses. All new works based on yours will carry the same license, so any derivatives will also allow commercial use.
    - **Public Domain (CC0)** - Use this universal tool if you are a holder of copyright or database rights, and you wish to waive all your interests in your work worldwide.

- **API** - The part of the discussion be defined (unfortunately) by the Oracle v Google Java API copyright legal battle, and in light of the ruling, I urge you to consider one of the more liberal Creative Common licenses.

    - **Attribution (CC BY)** - This license lets others distribute, remix, tweak, and build upon your work, even commercially, as long as they credit you for the original creation. This is the most accommodating of licenses offered.
    - **Attribution-ShareAlike (CC BY-SA)** - This license lets others remix, tweak, and build upon your work even for commercial purposes, as long as they credit you and license their new creations under the identical terms. This license is often compared to copyleft free and open source software licenses. All new works based on yours will carry the same license, so any derivatives will also allow commercial use.
    - **Public Domain (CC0)** - Use this universal tool if you are a holder of copyright or database rights, and you wish to waive all your interests in your work worldwide.

- **Client Code** - Separate from your server side code, you should make sure all of your client side code SDKs, PDKs, and starter kits have an open source license applied-o-remember you are asking them to potentially integrate this into their business operations. Again I recommend using Github's Choose a License to help you navigate this decision.

    - **Apache** - The Apache License is a free software license written by the Apache Software Foundation (ASF). The Apache License requires preservation of the copyright notice and disclaimer. Like other free software licenses, the license allows the user of the software the freedom to use the software for any purpose, to distribute it, to modify it, and to distribute modified versions of the software, under the terms of the license, without concern for royalties.
    - **GPL** - The GNU General Public License (GNU GPL or GPL) is the most widely used[6] free software license, which guarantees end users (individuals, organizations, companies) the freedoms to run, study, share (copy), and modify the software. Software that allows these rights is called free software and, if the software is copylefted, requires those rights to be retained.

    - **MIT** - The MIT License is a free software license originating at the Massachusetts Institute of Technology (MIT). It is a permissive free software license, meaning that it permits reuse within proprietary software provided all copies of the licensed software include a copy of the MIT License terms and the copyright notice.

Branding

With the programmatic, and seamless nature of API integration, loss of corporate and product branding control is a major concern. After security, branding is probably one of the most common inquiries when it comes to API operations. While there is no silver bullet to ensuring branding protection, there are a number of common approaches used by some of the largest, and most well known brands when it comes to API operations.

This is an exploration of some of the common building blocks used when it comes to branding resources, and requirements at the API level. Like a handful of the other layers of the API lifecycle, branding resources are not as widely available as they should be. This research looks to better map out what resources their are, identifying the common building blocks of healthy API branding.

- **General Considerations** - I just have one bucket for the branding area of my research. Like terms of service, and privacy, I am just getting going on this legal side of the API research, and it will evolve with time. There are the main areas of concern for branding at the moment.

    - **Use of Brand Name** - How should API consumers understand when it comes to using brand names in their tools, services, and any materials they produce to support their work.
    - **Use of Brand Logo** - How should API consumers understand when it comes to using logos in their tools, services, and any materials they produce to support their work.
    - **Use of Product Titles** - How should API consumers understand when it comes to using product titles in their tools, services, and any materials they produce to support their work.
    - **Image Assets** - What image assets are available to support and encourage branding efforts by API consumers? Could be logos, all the way to banners--anything image related that supports branding.
    - **Other Assets** - What other assets are available at your company that you use to help guide internal branding strategies, that maybe you could repurpose, and share with the developer community, helping them be more successful in representing your brand.
    - **Linking Requirements** - What are your linking requirements? Along with images and other assets, what URLs should be attributed to as part of the process. Make the instructions clear and simple, with all URLs easily available, throughout all assets, and branding related tooling.
    - **Naming Your Application** - What are the considerations when an API consumers is naming their application that is build on top of an API? Are there any reserved words, or things consumers should be considering?
    - **Branding Examples** - What are some existing examples of how brand can be used. Are there samples, or actual case studies of existing integrators that can show how branding is done properly, or maybe done improperly.
    - **Give Credit** - Explaining to API consumers that they should give credit where appropriate, and use common sense when it comes to integrations and giving proper credit to value generated via the API.

    - **Bring Value** - Explaining to API consumers that they should bring value to the platform, and not extract more value than they create. Demonstrating that this balance is needed, to make the API integration relationship work.

Discovery

Discovery and exploration of APIs, using simple desktop, or web HTTP clients, is becoming a common way to learn about what an API does, and potentially decreases integration time for API consumers. This growing breed of HTTP clients designed for API consumption, is also being used by API providers to better collaborate, communicate, and share information during the API design, and deployment process.

This is a much newer layer to the API sector, but is one that is fast growing, as well as colliding with other areas of research like API design, virtualization, and even deployment. This research could easily be absorbed into these other projects, but for now remains a single grouping of some of the common building blocks of these cloud-based, API clients, focused on making the process of providing, and consuming APIs is as efficient as possible.

- **Specification** - A machine readable specification designed to assist in the area of API discovery. This allows APIs, and their supporting operations to be described in a way that can ingested, and indexed by API search engines, and directories.

    - **APIs.json** - Use the APIs.json JSON specification for describing collections of APIs, allowing them to be indexed for inclusion in public and private search engines.

- **Discovery** - How APIs are being discovered across the current API landscape. How are APIs being found by developers, and application architects at all stages of development.

    - **API Directory** - API directories allow for the submissions and curation of APIs, then enable users to search and browse by keywords and tags, in many areas.
    - **API Hub** - API directories usually span many categories, where hubs often focus on specific area like telco, or healthcare, providing a single hub for discovering APIs.
    - **IDE Extension** - integrated development environment (IDE) extensions or addons that allow developers to quickly discover and understand API endpoints.
    - **API Explorer** - Sometimes also called consoles are designed to provide an interactive interface for developers to discover, explore, make calls and see responses in an interactive way.
    - **API Questions** - Providing a list of questions about API operations, using the api-questions-format, allowing common aspects to be indexed and discoverable via API search.

- **Directory** - Moving to actual examples of API directories that exist, providing static developers that API consumers can browse, and search by keywords within.

    - **ProgrammableWeb** - The original API directory created by John Musser in 2005, to provide a single, searchable directory of common public APIs.

    - **Mashape** - An API directory born out of the API management platform Mashape, providing a directory of Mashape clients APIs.

- **Business** - Business directories that allow for additional information related to APIs, as well as having APIs themselves, allowing for discovery of APIs from companies who list themselves in these directories.

    - **Crunchbase** - A directory of companies that is curated by Techcrunch, and allows an approach to listing companies who have APIs.
    - **AngelList** - Like Crunchbase, AngelList is a directory of companies, but also provides a way to showcase companies who have APIs.

- **Search** - What search engine solutions are available out there, that allow for API discovery, moving beyond just static API discovery.

Client

Immediately after discovery, you need the ability to explore an API. Even if it is immediately after you have an idea for an API. Modern APIs allow you to quickly mock a new, or explore an existing API, by putting in an endpoint, define authentication, and seeing what you get back. Modern HTTP clients allow you to discover, explore, save, organize, and collaborate around the APIs that matter.

In this research, client exists before we reach the point of IDE or SDK. At the client level, we are learning, and exploring, understanding the value delivered by any single, or group of APIs--before we commit. The client is the waiting room for API consumption, either as we are designing, defining, and deploying an API, or as we are just learning about what an API does, here are the building blocks that are in place.

- **Request Editor** - What tooling and services are available, which allow API consumers to make requests, manage the details of these requests, and be as efficient as possible when it comes to editing API requests.

    - **Request URL Editor** - Having full control to edit any part of the URL for any request being made through the client.
    - **Request Headers Editor** - Being able to add, edit, and remove any headers that will be sent with any request being made through the client.
    - **Cookies Manager** - Being able create, edit, and remove cookies that control any session of a request being made through the client.
    - **Request Method Manager** - Allowing for the use of all HTTP verbs, identify each method that is being made as part of a request through the client.
    - **Request Body Editor** - The ability to add, edit, or remove the entire body of a request being made through the client.

- **Authentication** - What are some of the common authentication methods available?

    - **Basic Auth** - Basic Auth is supported by an HTTP client for authentication as part of ay request being made.
    - **Digest Auth** - Digest Auth is supported by an HTTP client for authentication as part of ay request being made.
    - **OAuth 1.0** - OAuth 1.0a is supported by an HTTP client for authentication as part of ay request being made.
    - **OAuth 2.0** - OAuth 2.0 is supported by an HTTP client for authentication as part of ay request being made.

- **Environment** - What environments are available for isolation API integration within? Is there the ability to setup, maintain, and evolve environment meta information, and variables to support many different APIs.

    - **Separate Environments** - The HTTP client allows for establishing of separate environments to make requests to APIs in different modes.
    - **Saved Variables** - Allow for naming and saving of variables that can be used across API requests, in separate environments.

- **Response Viewer** - Am I given the ability to view responses, in different ways, as I make requests, and other aspects of API integration.

    - **Save Requests** - Allow individual requests to be saved, and run again in the future with the saved configuration.
    - **XML Viewer** - Provide a viewer for seeing the XML returned, and be able to copy, paste, save, and other features.
    - **RAW Viewer** - Provide a viewer for seeing the raw response returned, and be able to copy, paste, save, and other features.
    - **Search** - Allow the searching of results, by keyword and key phase, returning the point in results that match search.

- **Organization** - How am I able to organize API calls within my API client tooling and services. Can I organize them into folders, groups, or any other approach.

    - **Collections** - Allow for requests to be saved, and stored into collections that can be named, organized and managed in meaningful groups.
    - **Templates** - Provide starter templates of common public APIs, allowing HTTP client to be immediately put to use from common definitions.
    - **Clone Requests** - Enable the HTTP client to clone existing requests, and generate new collections or requests from existing definitions.
    - **Record** - Allow for any saved request to be recorded, for playback later.
    - **Replay** - Allow for the replay of any saved requests that has been run and recorded.
    - **Keyboard Shortcuts** - Provide keyboard shortcuts, that make working with HTTP client requests easier.
    - **History** - Provide a historical listing of API requests that have been made, allowing the history to be searched, reviewed, and managed.

    - **Teams** - Provide team environments, allowing users to collaborate, share and work across API requests, and collections.

- **Import / Export** - Import and export of common API definitions is becoming more commonplace in almost every area of the API lifecycle. There are the common ones being employed today.

    - **Import Swagger** - Allow for the importing of API requests using the Swagger API definition format.
    - **Import API Blueprint** - Allow for the importing of API requests using the API Blueprint API definition format.
    - **Import RAML** - Allow for the importing of API requests using the RAML API definition format.
    - **Import Postman** - Allow for the importing of API requests using the Postman API definition format.
    - **Export Postman** - Allow for the exporting of API requests using the Postman API definition format.
    - **Export Swagger** - Allow for the exporting of API requests using the Swagger API definition format.
    - **Export API Blueprint** - Allow for the exporting of API requests using the API Blueprint API definition format.
    - **Export RAML** - Allow for the exporting of API requests using the RAML API definition format.

- **Tooling** - What other tooling is available, to make API integration, and clients more effective, and usable by API developers. Some of these areas may touch on other stops along the API lifecycle.

    - **Proxy** - Provide a proxy that can be used when making requests via the HTTP client.
    - **Extensions** - Allowing the extension of an HTTP client using a standardize plugin interface.

Integrated Development Environment (IDE)

Another overlapping area of research with API clients, is the integrated development environment, or IDE. There are a wide variety of IDEs out there, but a handful of them are taking notice of the growth in API development, and providing more resources, services, and tools focused on APIs. Like almost every other area of this research, some IDE environments are also employing APIs, to allow for more automation, in this DevOps, cloud environment we are finding ourselves in.

There are a number of building blocks emerging, that are enabling other areas of the API lifecycle, including discovery tooling, to being able to virtualize, and monitor APIs directly from the IDE. This is another fairly young section of my research, but I am already seeing signs of significant growth, and attention to APIs, within the IDE development space. What are some of the common elements present? This is a checklist to explore some of what is emerging.

- **Core Elements** - What are some of the general ways that IDEs and APIs are working together. APIs at the IDE level are about engaging with almost every other stop along the API lifecycle.

    - **Workspace** - Workspace definition tooling and services, allowing APIs to be an integrated part of the IDE workspaces, either by default or custom addition.
    - **Project** - Allowing developers to break up API development into specific projects, providing well defined, project focused organization for API integration.
    - **Container** - Are there options to be able to containerize individual API integrations, isolating them into specific virtualized instances.
    - **Resources** - What API focused resources are available within an IDE environment. How do developers find documentation, and other resources that support them in their integrations.
    - **Analytics** - What analytics are available to support API integrations, during development? Is the ability to track and measure API related integrations during design and development time.
    - **Environment** - Is there the option for establishing separate environments within an IDE. Are developers able to segment and group their API integrations into multiple environments.
    - **Github** - Using Github as a source for code, templates, definitions, and other essential building blocks of the API development process, in a modern IDE.
    - **Editor(s)** - Are there options for multiple editors, depending on media or file type, or possible set per project and environment levels, within the IDE.
    - **Plugins** - Does the IDE environment allow for plugins, and extending the environment? Plugins allow for the introduction of 3rd party tools and services into the IDE workspace.
    - **Autocomplete** - One of the hallmark features of an IDE, is autocomplete for specific languages and frameworks. In this scenario autocomplete is focused on dictionaries created by API definitions, meant for specific integrations.
    - **Customize** - Are there options for customizing the IDE, from look and feel, to the functionality.

Software Development Kits (SDK)

It is common practice to provide developers with a wide as possible selection of software development kits (SDKs), that will reduce the resources needed during a successful integration. While not everyone is in agreement of the need for SDKs, or the best possible practices for SDK management, SDKs are commonplace, and should be consider as part of planning of any APIs. SDKs come in many shapes and sizes, and have many different requirements for various languages, and platforms.

With this area of research, I am looking to identify the common patterns for SDK development and maintenance. There are different approaches in different industries, and increasingly for more device based API implementations. The best we can do, like other areas of the API lifecycle, is aggregate the common patterns, so we can use in the decision making process. Considering the common approaches to SDKs is an essential, while the actual execution may not be something every provider should look to do, unless they have the resources needed to do right.

- **General SDKs** - What SDK generation capabilities are available? These SDKs might be hand crafted, or auto generated, but should be available in a variety of languages, encouraging the jumpstarting of integrations by a wide as possible audience.

    - **C Sharp** - An SDK available in the C Sharp / C# programming language, with consideration specifically tailored for the language.
    - **Objective-C** - An SDK available in the Objective-C programming language, with consideration specifically tailored for the language.

    - **Java for Android** - An SDK available in the Java programming language, with consideration specifically tailored for Android.
    - **Java for JVM** - An SDK available in the Java programming language, with consideration specifically tailored for the language.
    - **PHP** - An SDK available in the PHP programming language, with consideration specifically tailored for the language.
    - **Python** - An SDK available in the Python programming language, with consideration specifically tailored for the language.
    - **AngularJS** - An SDK available in the JavaScript programming language, with consideration specifically tailored for use in the Angular framework.
    - **Ruby** - An SDK available in the Ruby programming language, with consideration specifically tailored for the language.
    - **Go** - An SDK available in the Go programming language, with consideration specifically tailored for the language.
    - **Swift** - An SDK available in the Swift programming language, with consideration specifically tailored for the language.

- **Import / Export** - Like most other stops along the modern API life cycle, the SDK portion is being defined by a range of common API definition formats, which allow APIs to be imported and exported as needed.

    - **Import OpenAPI Spec** - Allow for importing of API definitions using OpenAPI specification format, for generation of SDKs.
    - **Import API Blueprint** - Allow for importing of API definitions using API Blueprint format, for generation of SDKs.
    - **Import Postman** - Allow for importing of API definitions using Postman format, for generation of SDKs.

- **Discovery** - How are SDKs discovered by developers during development? What are the considerations for making sure existing SDK efforts get found.

    - **List SDK** - Are SDKs listed in an easy to find location, where developers can browse them easily? A simple list goes a long way in making them found.
    - **Search SDK** - Is there the ability to search for SDKs, using a single index dedicated to them, or possibly using Github's already available search mechanisms.
    - **Browse SDK** - Is there a browseable directory of SDKs, that developers can use to find the SDK that they need? Potentially giving more information than just a listing.

- **Mobile Management** - There are many overlaps with mobile in the regular SDK portion of this research, but some providers are publishing more resources specifically dedicated to the support of mobile integrations.

    - **Mobile Overview** - An overview page, dedicated to mobile application development, giving equal time to each platform, including iOS, Android, and Windows.

    - **iOS SDK** - Providing mobile focused SDKs for developers to build IOS mobile applications on top of an API.
    - **HTML5** - Providing mobile focused SDKs for developers to build HTML5 mobile applications on top of an API.
    - **Appery.io** - Providing mobile focused SDKs for developers to build mobile applications on top of an API, using Appery.io.
    - **Windows Mobile SDK** - Providing mobile focused SDKs for developers to build Windows mobile applications on top of an API.

- **Code - Platform Development Kits (PDK)** - Beyond language specific SDKs, there are a growing number of platforms who make platform specific SDKs available, assisting developers in successfully integrating with existing 3rd party platforms.

    - **Wordpress** - Providing ready to go WordPress integration, allowing developers, and sometimes even non-developers to immediately put an API to use through their WordPress site(s).
    - **Heroku** - Providing ready to go Heroku integration, allowing developers, and sometimes even non-developers to immediately put an API to use through their Heroku platform account.

Embeddable

This area of research focuses on embeddable, usually JavaScript driven code goodies that help drive platform adoption, integrations, syndication, and many other essential aspects of API operations. Embeddable items can take on different forms, such as widgets, or even as a badge, or button. There are many well known examples of API driven embeddables, such as the Twitter share button, or the Facebook Connect login--the one thing they all have in common is API.

Some embeddables are about simple, often formats driven JavaScript goodies, while more sophisticated approaches may provide tooling, engines, and APIs specifically for embedding, and syndication. This area could easily be considered a part of the SDK area, but embeddables often transcend standard integration conversations, and can be used as part of branding, evangelism, and other critical areas of API operations.

- **Embed Formats** - What are the common embeddable formats available to API providers? There are two leading open formats available, that I track on.

    - **Open Graph Protocol** - The Open Graph protocol enables any web page to become a rich object in a social graph, by embedding of simple tags in the page header that describe objects.
    - **oEmbed** - Format for allowing an embedded representation of a URL on third party sites to display embedded content (such as photos or videos) when a user posts a link to that resource.

- **Embeddable Tools** - There are some embeddable tooling that is being employed by API providers to make content, data, and other resources available. These are a handful of the approaches being used

out there.

    - **Widgets** - Widgets are highly functional, API driven JavaScript tools that provide portable applications that can be embedded on any website or application.
    - **Buttons** - Buttons are shareable snippets of code that often share, syndicate or trigger a variety of events that use APIs as their source.
    - **Bookmarklet** - A bookmarklet that can be added to the toolbar of a browser, and quickly push data to an API.

- **Embed Engines** - Beyond simple tooling, there are more robust embeddable engines that are used in support of API operations, and integrations. These are a couple of the embed engines being employed right now.

Webhooks

A webhook allows API consumers to subscribe to specific events that occur via API operations, pushing out notifications, or even content and data to consumers--making API operations a two-way street. Beyond making operations multi-directional, they can alleviate the use of platform resources, unnecessarily. Webhooks play several important roles in platform health, scalability, and even communication.

There are no specific webhook standards, but we can track on the common patterns employed by API providers, and the handful of service providers who provide webhook services and tooling. This research looks to aggregate the best practices, and approaches when it comes to webhooks, and provide a listing of building blocks that can considered, and applied as part of the API lifecycle.

- **Core** - This is an exploration of the core aspects of Webhooks operations, providing a common set of building blocks that can be considered as part of API operations. Webhooks provide a two way street that benefit both provider and consumer, and can help make platforms more efficient.

    - **URL** - Tooling for managing the URL of a Webhook.
    - **Payload** - Editing what goes into the payload of the Webhook.
    - **Event** - Selecting which events triggers a Webhook.
    - **Content Type** - Selecting the content type for a Webhooks being submitted.

- **Inbound** - What tooling and services are available to manage the inbox targeting of platform Webhooks.

    - **Webhooks Targets** - Providing targets where Webhooks can post information, with targets then providing other services once received.

- **Outbound** - Webhooks tooling and services are available to manage the outbound targeting of platform webhooks.
    - **Multiple Destinations** - Sending of received Webhooks to multiple destinations once received, providing routing opportunities for Webhooks.
    - **CRON Jobs** - Running code that can be used as webhook target, also as schedule CRON jobs allowing them to trigger on schedule instead of just events.
- **Operations** - What Webhooks tooling is available to manage the overall operations, giving more visibility into Webhook exchanges.
    - **Analytics** - Providing analytics on webhooks activity, triggers, usage, and other aspects of operations.
    - **Emails** - Trigger the sending of emails when webhooks are triggered, received, or other aspects of operations.
    - **Logging** - The logging of all activity generated via webhook activity.
    - **Alerts** - Triggering of alerts based upon events, schedule, and other webhook operations.
- **3rd Party Integration** - What 3rd party integration options are available for Webhooks, from defining to running, and importing and exporting.
    - **Github** - Providing Github integration for webhook targets, scripts, and other elements of operation.

Monetization

It takes money to make APIs operate as expected. There are many different ways that API providers are monetizing their digital resources. Leading providers like Amazon has this figured this out at new levels, while other newer players are still experimenting, hoping to find the sweet spot of what it costs to operate, and what API consumers will pay. API monetization is not a one-way discussion, and focused on making money, they can be used as revenue drivers, and even paying out revenue share with partners.

API monetization isn't just about what you charge for API access. For a holistic API monetization strategy you must start at the beginning and look at what it costs to acquire, develop, operate, and evolve an API. In the more mature layers of the space like payments, messaging, and cloud computing, the monetization approaches are getting more proven, while we are still mapping out, and exploring new ways in which resources are digitized, and identifying the common employed, and even determining what the best, and most successful patterns might be.

- **Acquisition** - What are the acquisition considerations for API monetization--what costs go into acquiring everything there is needed for API operations.

    - **Discover** - What did I spent to find this. I may have had to buy someone dinner or beer to find, as well as time on the Internet searching, and connecting the dots.
    - **Negotiate** - What time do I have in actually getting access to something. Most of the time it's on the Internet, and other times it requires travel, and meeting with folks.
    - **Licensing** - There is a chance I would license a database from a company or institution, so I want to have this option in here. Even if this is open source, I want the license referenced as part of acquisition.
    - **Purchase** - Also the chance I may buy a database from someone outright, or pay them to put the database together, resulting in one-time fee, which I'm going to call "purchase".

- **Development** - What are the acquisition considerations for API monetization--what costs go into developing everything there is needed for the platform to operate.

    - **Investment** - Who put up the money to support the development of this API resource? Was it internal, or did we have to take external investment.
    - **Grant** - Was the development of this API rolled up in a grant, or specifically a grant for its development. Covering costs involved.
    - **Normalization** - What does it take me to cleanup, and normalize a dataset, or across content. This is usually the busy janitorial work necessary.
    - **Design** - What does it take me to generate a Swagger and API Blueprint, something that isn't just auto-generated, but also has the required hand polish it will require.
    - **Database** - How much work am I putting into setting up the database. A lot of this I can automate, but there is always a setup cost involved.
    - **Server** - Defining the amount of work I put into setting up, and configuring the server to run a new API, including where it goes in my overall operations plan.
    - **Coding** - How much work to I put into actually coding an API. I use the Slim PHP framework, and using automation scripts I can generate 75% of it usually, but there is always finish work.
    - **DNS** - What was the overhead in me defining, and configuring the DNS for any API, setting up endpoint domain, as well as potentially a portal to house operations.

- **Operation** - What are the acquisition considerations for API monetization--what costs go into operating everything there is needed for API to function daily.

    - **Definition** - How much resources am I applying to creating and maintaining APIs.json, Swagger, and API Blueprint definitions for my APIs.
    - **Compute** - What percentage of my AWS compute is dedicated to an API. Flat percentage of the server it's one until usage history exists.
    - **Storage** - How much on disk storage am I using to operate an API? Could fluctuate from month to month, and exponentially increase for some.
    - **Bandwidth** - How much bandwidth in / out is an API using to get the job done.
    - **Management** - What percentage of API management resources is dedicated to the API. A flat percentage of API management overhead until usage history exists.
    - **Code** - What does it cost me to maintain my code samples, libraries, and SDKs for each individual API, or possibly across multiple APIs and endpoints.
    - **Evangelism** - How much energy do I put into evangelizing any single API? Did I write a blog post, or am I'm buying Twitter or Google Ads? How is the word getting out?

    - **Monitoring** - What percentage of the API monitoring, testing, and performance service budget is dedicated to this API. How large is surface area for monitoring?
    - **Security** - What does it cost for me to secure a single API, as part of the larger overall operations? Does internal resource spend time, or is this a 3rd party service.

- **Direct Value** - What are the units of currency the platform uses. What are the individual value units applied to each API, and how are things calculated. Most like this is done in dollars, or euros, but other units are emerging as well.

    - **API Value** - Each API will have its own credit rate set, where some will be one credit, while others may be 100 credits to make a single call, it can be defined by API or a specific endpoint.
    - **Limits** - The daily allowed credit limit will be determined by the access level tier is registered at, starting with daily free public access to retail, trusted, and potentially custom tiers.
    - **Usage** - How many credits does any one user use during a day, week, or month, across all APIs. When each API is used, it will apply the defined credit value for the single API call.
    - **Incentive** - How can the platform give credits as an incentive for use, or even pay credits for writing to certain APIs, and enriching the system, or driving traffic.
    - **Volume** - Are there volume discounts for buying of credits, allowing consumers to purchase credits in bulk, when they need to and apply when they desire.
    - **Value** - What is the value of an API resource? Often you will be just making this up, but at the least you should be able to articulate in some way.
    - **Revenue** - What revenue opportunities for the ecosystem around an API and its operation, sharing in the money made from platform operations.

- **Indirect Value** - Beyond the obvious, APIs are generating a lot of value for platform providers, and consumers. What are some of the common ways to look at indirect value generation.

    - **Marketing Vehicle** - Having an API is cool these days, and some APIs are worth just having for the PR value, and extending the overall brand of the platform.
    - **Traffic Generation** - The API exists solely for distributing links to web and mobile applications, driving traffic to specific properties - is this tracked as part of other analytics?
    - **Brand Awareness** - Applying a brand strategy, and using the API to incentivize publishers to extend the reach of the brand and ultimately the platform - can we quantify this?
    - **Data & Content Acquisition** - Using the API, and the applications built on top as part of a larger data and content acquisition strategy--can we quantify this?

- **Partner Revenue Generation** - How is revenue being generated specifically for partners? There are a number of common approaches to revenue sharing with partner tiers of API access.

    - **Link Affiliate** - What revenue is generated and paid out via links that are made available via the API, with potentially externally affiliate links embedded.
    - **Revenue Share** - What portion API revenue is paid out to potential re-sellers who drive API usage. Revenue is percentage of overall credit purchases / usage.

    - **Credits to Bill** - All revenue is paid in credits on account, and user can decide to get buy out of credits at any time, or possibly use in other aspects of system operation.

- **Reporting** - How is revenue from APIs tracked, organized, and reported on. API value should be quantified in as many ways as possible, and shared accordingly to make sense of revenue generated.

    - **Timeframe** - How much revenue is being brought in on a daily, weekly, monthly, quarterly, or annual basis for an API and all of its endpoints.
    - **Users** - How much revenue is being brought in on a monthly basis for a specific user, for an API and all of its endpoints.
    - **Plans** - Which plans generate most usage and revenue? I should be applying just as easily to aspects of platform / internal usage as well. Better understanding value generated across all plan levels.
    - **Affiliate Revenue** - What was generated from affiliate links made available via APIs, minus what percentage was paid out to API consumers.
    - **Advertising Revenue** - What advertising revenue was derived from web or mobile application traffic resulting from the API, minus whatever was paid out as rev share to API consumers.

Plans

API monetization is about all the thinking that actually goes into how to operate, and generate revenue via an API platform. API plans are how you actually execute on this vision, and maximize spend and revenue within the API monetization framework you have established. API plans are the guide to API service composition, which is the governor for all API consumption via any modern platform, determining who gets access to what resources, how much they get, and what they will pay for this access, or even get paid.

It is common to see just a handful of plans published via any modern API platform, but in reality there is often many more layers of access, as well as all plans potentially possessing a complex variety of variables and elements that govern access to digital and physical resources made available via APIs. There is not any universal way of assembling API plans, but there are many established models we can follow, and this research is about breaking down the elements of these leading models, in a way that we can then choose which best apply to our overall API operational objectives.

- **Elements** - These are the key elements of API plans that I have gathered from across hundreds of API providers. These elements can be associated with specific plans that are available, but they do not have to, and I often use them to generally describe the plans, or perceived plans behind API operations.

    - **Overview** - Providing a single landing page, available at a simple URL, with all the information about plans available as part of platform operations. There may be more than one plan overview page available, for example one for SaaS side of offering, one for developers, and another possibly for enterprise or partner consumers.
    - **Private** - Is this plan a private one, available only to a limited audience? While the landing page, or overall portal might be publicly available, the API access itself requires approval, or

existing account access before you can get more details. Private APIs are more common than public APIs, but you should think about the pros / cons of keeping things private.

- **Public** - Is this a publicly available API operation, something that may only apply to some plans? While not all aspects of API operations will be 100% publicly available, there are some elements that anyone from the public can gain access to, even if it is rate limited in some way.
- **Free** - Is this a free resource, or set of resources, something that may not apply to all plans. Free should be considered even if it is part of a trial, demo, or just a limited level of access by time other metric. Free should be a way to incentivize users to higher level of access, and not damage potential revenue.
- **Volume** - Volume pricing levels are available, with plans based entirely on volume level access to resources. Some platforms operate entirely on volume levels of API access, and levels should be crafted with end consumers in mind, and how they will be consumed, and at what levels.
- **Subscription** - There is a regular, recurring element to accessing a resource, as part of a larger subscription part of, or separated from plans. In addition to the regular usage, a subscription plan usually continues, and has costs associated with it even if there are no calls against the API made.
- **Commercial** - There is a commercial licensing element to a set of resources, which require additional business or legal requirements. This element may not be cost associated, and just stipulate there are commercial opportunities around an API. These elements may require requesting access, or additional steps before they can become available.
- **Non-Commercial** - The platform allows for non-commercial access to resources, which usually means there is additional commercial requirements. Allowing non-profit organizations, researchers, and other potential API consumers the ability to validate their identity, and achieve reduced, or free levels of access can benefit the overall platform reach, and potential marketing engine.
- **Internal** - A set of resources are tailored specifically for internal consumption rather than for public or partner level access. Internal APIs, like public ones will usually have a portal available at an Internet URL, but will require additional access before you can see anything related to the API.
- **Partner** - There is a focus on partner access to resources, resulting in separate access layers, and features dedicated to partner level consumption. While anyone accessing APIs can be considered a partner, these levels of access usually require much more vetting, making sure business objectives are in alignment, and legal obligations are met. Keeping partner access layers as transparent as possible helps keep harmony with lower levels of public, or even internal API access.
- **Features** - Additional features, usually key / value pairs that can be used to describe addition elements of API access and consumption. The purpose of this research is to identify some of the common elements of how API providers plan their operations, and there will be a long tale of features that I do not capture--these are features.
- **Marketing** - Is a big part of API operations about driving marketing efforts for other aspects of business operations. Direct API access is not always the primary motivation for having an API, especially a public API. There are a number of ways to increase attention to products, services, as well as the overall corporate engine using APIs. Acknowledging this across API planning is important to understanding the value generated by API consumption, beyond direct elements.
- **Branding** - Is there a strong brand aspect to API operations, with clear guidelines, resources, and a presence for the company's brand. It is common to require API consumers to provide

brand attribution on their site, and sometimes on web or mobile applications, as part of the API consumption contract. Branding should be reflected as an element for all public APIs, with details about different requirements for different plan levels.

- **Traffic** - Is a primary focus of an API platform centered around driving traffic to a web or mobile presence. If there is a solid business model around users accessing content or data, driving traffic through an API, without charging API consumers can be a successful approach. How can API consumers be incentivized to drive more traffic to web and mobile properties, by extending the reach of the platform through APIs.
- **Content** - One of the primary objectives for API resources existing is in support of acquiring data and content that fuels platform operations. This is a very common element of successful API platforms that are operating today, such as Crunchbase or Angellist, or even Twitter and Tumblr. These are all examples of platforms where content acquisition is a core element of their API plans.
- **Products** - An API primarily exists to support the existence, or drive the sales of another product, or product line. I separate out devices from this grouping, because they require separate attention. This could describe entire supply chain integration using APIs, all the way to affiliate or reseller style systems like used by Amazon. There are a number of ways products can be sold, by putting APIs to work.
- **Services** - API resources exist to support other related services, and act as value-add to the primary business focus. This is somewhat duplicate to SaaS or PaaS elements, but more likely is about the delivery of classic services like Uber, TaskRabbit, and other sharing economy solutions are focused on today. How is an API driving the demand of other services, that may not be software based, and bridge into the physical world.
- **API** - There are APIs available to access all or part of the plans, pricing, or limitations around an API platform operations. The API platforms I track on that have been around the longest, and are the most mature, have APIs for consumers to use, that gives access to plan information, rate limits, pricing, and other aspects of API operations. I include this as essential, because it will be something every API provider will need to complete in the future.

- **Timeframes** - The consumption of API resources is often measured within timeframes, in addition to the wide number of other metrics that can be applied. Having meaningful timeframes defined for evaluating how APIs are consumed, and using as part of overall planning, when it comes to all aspects, ranging from rate limits to billing.

    - **Seconds** - Managing, guiding, and restricting plan entries in seconds. This is a common timeframe for considering rate limits, and judging the overall volume requirements of different users. Availability in seconds is often directly linked to compute resources being applied as part of operations, and tie in with overall availability.
    - **Hourly** - Managing, guiding, and restricting plan entries in hours. While there may be rate limits at the hourly level, this timeframe is more applied to resources that are on-demand and ephemeral, and can be consumed as needed, in a utility style that is becoming common way to plan API access.
    - **Daily** - Managing, guiding, and restricting plan entries in days. Like seconds and minutes this is a common timeframe for considering rate limits, and judging the overall volume requirements of different users. Availability in seconds is often directly linked to compute resources being applied as part of operations, and tie in with overall availability.

    - **Monthly** - Managing, guiding, and restricting plan entries in months. Like weekly, this timeframe is more used to organizing billing and support cycles, organizing resource usage and services rendered within the weekly time period, and aligning billing, and other aspects to this timeframe.

- **Metrics** - Beyond the overall elements, and timeframes to consider, what are the specific metrics that are being applied to overall API operations, as well as individual plans and access tiers. Depending on the resource, there are a number of metrics being used across the API space, by leading API providers. This layer of the journey is meant to walk through the metrics you will want to consider in your API journey, allowing to cherry pick the ones that are most import to you. Not all metrics apply in all situations, but they are the building blocks of good API plans.

    - **Calls** - The most fundamental metric for APIs, the API call.
    - **Value** - What is the value of an API or endpoint, is it simple 1, or maybe 10, allowing for multiple value settings.

- **Limits** - What are limitations and constraints applied as part of the API planning operations. How are these crafted, applied, and reported upon.

    - **Overview** - A single page which provides an overview of limits that are in place on API operations.

Partners

API operations are all about access to resources, internal, and with trusted partners. The biggest myth perpetuated about APIs, is that you just give away resources online, when in reality they give you very granular level control over the resources that are being made available. Leading API providers have well thought out partner tiers of access, augmenting the service composition, that modern approaches to APIs afford us. These leading API partner programs provide us with a possible blueprint that can be considered as part of API operations.

Operating APIs, that make valuable resources available via the Internet, offers new opportunities for business development, in a much more self-service, scalable way. Approaches to API partner programs provide a viable way to qualify, engage, and even showcase partners who prove to be a valuable part of API operations. Partners are essential to the health of API ecosystems, and they are essential to business relationships in a global, online, digital business environment.

- **Program Details** - The communication around partner levels of access is critical to overall health and balance with other tiers of access. Providing as much detail for partners, but also potentially other levels of access is important. Here are a few of the building blocks employed to help manage partner details.

    - **Landing Page** - An official landing page for the partner program.

    - **Program Details** - Short, concise information about the program.
    - **Program Requirements** - Information about what the details of the program are.
    - **Program Levels** - Details about what the different levels of the partner program are.

- **Partner Showcase** - For many leading API providers, showcasing partners is an important aspect of platform partner operations. Showcasing who is involved, provides transparency, as well as incentive for other platform users, both external and internal.

    - **List of Partners** - A list of partners, usually name and logo, maybe with some description.
    - **Partner Search** - A keyword search for discovering partners.

- **Partner Program** - Beyond the details, and showcasing partners, what are the core elements of the program itself? Here are some of the common elements put to work as part of partner programs in operation across the space.

    - **Application** - The actual application for becoming a partner.
    - **Private Portal** - A private portal for partners to login to.
    - **Certification** - Official certification showing that a partner is officially approved and vetted.

- **API** - Like many other aspects of the API lifecycle, there are increasingly APIs available for managing aspects of partner access, program details, and overall partner program operations. These are some of the common building blocks being applied using APIs for partner tiers.

    - **Quota Increase** - Increasing the rate limit for existing APIs.
    - **Additional APIs** - Provide access to APIs that are only accessible to their partner tiers.
    - **Read / Write APIs** - Access to not only read, but also write, update, and possibly delete access to aPIs.
    - **Program API** - Providing an API that opens up access to all or part of partner program operations, providing programmatic access and transparency to the program. Like most other lines along the API lifecycle, partner programs should have APIs of their own.

- **Early Access** - Partner programs are often to give preferred access, to the trusted levels of partner access. This early access takes on many forms.

    - **Early Communication** - Allow for partners to get early access to platform communications.
    - **Early Opportunities** - Access to early platform opportunities meant just for partners.
    - **Beta Access** - Allow for beta access to new platform products.

- **Legal** - Beyond the usual terms of service, privacy policy, and legal aspects of platform operations, the partner program will often have their own set of legal constraints and protections.

    - **Agreement** - The legaleze for the partner program agreement.

- **Marketing Activities** - Partners may also enjoy special marketing activities, only available to these higher, more trusted tiers of access. These are some of the common marketing activities being applied across the space.

    - **Blog Posts** - Provide blog posts for partners to take advantage of one time or recurring.
    - **Press Release** - Provide press releases for new partners, and possibly recurring for other milestones.
    - **Twitter Post** - Post updates to the platforms Twitter account.

- **Support** - Partners may also enjoy specific support opportunities, allowing them to get more access to platform support resources, via their partner level access.

    - **Office Hours** - Provide virtual open office hours just for partners.
    - **Training** - Offer direct training opportunities that is designed just for partners.

- **Content** - What kind of content relationships can be established as part of partnership activities. Content generated from existing, successful relationships, can be a big driver in forming new partners, as well as keeping existing ones healthy.

    - **Quotes** - Allow partners to provide quotes that can be published to relevant properties.
    - **Testimonials** - Have partners provide testimonials that get published to relevant sites.
    - **Use of Logo** - Allow partners to use the platform logo, or special partner platform logo.

Evangelism

APIs do little good, if nobody knows about them. The latest wave of modern API providers have established some very proven practices when it comes to evangelizing APIs, and making sure internal groups, partners, and even the public are more aware that APIs exist, and the value they bring to the table. While not every API will need dedicated resources for evangelizing for APIs, but it should be an essential consideration for every API, in some form.

A common misconception about API evangelism or advocacy is that it is in outward facing endeavor, when it reality it should be applied internally, amongst partner groups, as well as the public, in equal parts that make sense in alignment with overall API goals. The most common reason APIs fail, is they lose internal support, and while it can be difficult to bring much needed attention to the fact that APIs exist, it might be even more difficult to keep management convinced an API should have the budget it needs to be successful. Carefully weight the best patterns to emulate as part of any API evangelism effort.

- **Goals** - What are the core goals of the API operation? These need to be precise, measurable, and obtainable goals. While there may be unique ones to your situation, these are some of the common ones I see.

    - **Growth in New Users** - Growing the number of new signups for an API platform, increasing the number of new consumers, and potentially establishing as an active user.
    - **Growth in Existing User API Usage** - Growing the number of existing users who are actively using an API platform, increasing the number of APIs they use, and the number of calls they make.
    - **Brand Awareness** - A consistent strategy and set of resources for extending the brand reach for any API platform, by providing a set of instructions and assets for consumers to use.

- **Developer Outreach** - Reaching out to developers is essential, not just to attract them as new users, but after they've registered, and as they are putting to the platform to work. Do not confuse sales with outreach, and make sure it is meaningful engagement.

    - **Fresh Engagement** - Emailing new developers who have registered weekly to see what their immediate needs are, while their registration is fresh in their minds.
    - **Active User Engagement** - Where we reach out to existing, active users of your API and find out what they need, profiling them via Twitter, Facebook, LinkedIn and pulling any profile information to grade and categorize each developer.
    - **Historical Engagement** - Emailing historical active and / or inactive developers to engage and understand what their needs are and would make them active or increase activity.
    - **Social Engagement** - When emailing any developer, using Rapportive we establish any URL, Twitter, Facebook or LinkedIn profile they have and reach out via these networks.

- **Blogging** - How does blogging occur via the platform? What approaches are being used to generate, produce, and syndicate stories, keeping a regular stream of information flowing from the platform.

    - **Projects** - Establishing of editorial assembly line of technical projects that can feed blog stories, how-tos, samples and Github libraries.
    - **Stories** - Writing, editing and posting of stories derived from projects, with SEO and API area support by design.
    - **Syndication** - Syndication to Tumblr, Blogger and other relevant blogging sites available online.

- **Landscape Analysis** - Every API operates within a specific space, and understanding the landscape of the space is very important to the health and effectiveness of evangelism efforts.

    - **Competition Monitoring** - Evaluation of the regular activity of competitors, cultivating an ever growing list of who they are and what they are up to.
    - **Industry Monitoring** - Evaluation of relevant topics and trends of overall industry an API is operating within.
    - **Keywords** - Established a list of keywords to use when searching for topics at search engines, QA, forums, social bookmarking and social networks.
    - **City Targeting** - Target specific markets in cities that make sense to an overall API evangelism strategy.

- **Forum Management** - Forums play a big role in the self-service, and ecosystem nature of API operations. Forums can be within a platform, as well as on existing public forums. These are some of the considerations with forums when it comes to evangelism.

    - **Forum Conversations** - Responding to forum activity in a timely and engaging way, and conducting general janitorial work around the community.
    - **Forum Posting** - Generating forum activity by anonymous posters or officially with common problems and questions faced throughout community as well as open landscape.
    - **Stories** - Deriving of stories for blog derived from forum activity, and the actual needs of developers to be crafted and published as part of overall editorial strategy.

- **Support** - What role does support play in the overall evangelism workflow. Evangelism is not just about marketing and sales, and much of the tone of evangelism gets set by a common set of support elements.

    - **Email Coordination** - Regular coordination, relaying of support issues with central support process.
    - **Email Needs Tracking** - Deriving of common support requests to drive API and API area roadmap.

- **GitHub Management** - Github plays a central role in many areas of the API life cycle, but the social nature of Github lends itself well to evangelism efforts. Here are some of the common elements I am seeing.

    - **Github Repository** - Managing of code sample Gists, official code libraries and any samples, starter kits or other code samples generated through projects.
    - **Github Relationship** - Managing of followers, forks, downloads and other potential relationships via Github.

- **QA Management** - Question and Answer (Q&A) sites and forums play a big role in the developer community. Here are some of the considerations when it comes to QA management for evangelism.

    - **QA Profile Maintenance** - Ensure that profile on Quora and Stack Overflow are active.
    - **Stack Exchange** - Regular trolling of Stack Exchange / Stack Overflow and responding to relevant topics or industry related questions.
    - **Quora** - Regular trolling of Quora and responding to relevant [Client Name] or industry related questions.

- **Social Management** - Social media plays a big role in business operations, and are just as critical to API evangelism efforts. Here are some of the common ways API providers are using social services to engage consumers.

    - **LinkedIn** - Setup of new API specific LinkedIn profile page who will follow developers and other relevant users for engagement. Posting of all API evangelism activities.
    - **Twitter** - Setup of new API specific Twitter account, and the tweeting of all API evangelism activity, relevant industry landscape activity, discover new followers and engage with followers.

- **Social Bookmarking** - Beyond just social networks, some social bookmarking sites are important to the API evangelism workflow. Here are some of the social bookmarking sites in use today.

    - **Product Hunt** - Listing of new API offerings via the Product Hunt social product and service platform, showcasing any new resources as they are made available.
    - **Hacker News** - Social bookmarking of all relevant API evangelism activities as well as relevant industry landscape topics to Hacker News, to keep a fair and balanced profile, as well as network and user engagement.
    - **Reddit** - Social bookmarking of all relevant API evangelism activities as well as relevant industry landscape topics to Reddit, to keep a fair and balanced profile, as well as network and user engagement.

- **Events** - Events are the in person face of any API operations. While much work can be done in an online environment, making sure you are available at events where API consumers will be. There are a number of proven events, that work for API evangelism.

    - **Hackathons** - What hackathons are coming up in 30, 90, 120 days? Which would should be sponsored, attended, etc.
    - **Meetups** - What are the best Meetups in target cities? Are there different formats that would best meet our goals? Are there any sponsorship or speaking opportunities?
    - **Conferences** - What are the top conferences occurring that we can participate in or attend--pay attention to call for papers of relevant industry events.

- **Reporting** - How are evangelism activities being reported upon. What are some of the common approaches to reporting on API evangelism efforts.

    - **Activity By Group** - Summary and highlights from weekly activity within the each area of API evangelism strategy.
    - **New Registrations** - Historical and weekly accounting of new developer registrations across APIs.
    - **Volume of Calls** - Historical and weekly accounting of API calls per API.

- **Internal** - Evangelism is not just an external thing. How is the platform being evangelized internally, even for publicly available APIs. Internal evangelism is very important for maintaining trust, and the ability to fund necessary platform resources.

    - **Storytelling** - Telling stories of an API isn't just something you do externally, what stores need to be told internally to make sure an API initiative is successful.

    - **Participation** - It is very healthy to include other people from across the company in API operations. How can we include people from other teams in API evangelism efforts.
    - **Reporting** - Sometimes providing regular numbers and reports to key players internally can help keep operations running smooth. What reports can we produce?

- **Developer Area** - What does the developer area look like for evangelism activities? What tools are available to help platform operators engage, and evangelize to consumers. Here are some of the common approaches.

    - **Getting Started** - Evolution of getting started page when relevant based upon API and network changes, and relevant developer feedback.
    - **Documentation** - Enhancement and addition to API documentation based upon project development and active developer engagement and feedback
    - **Code Samples** - Establish new code samples that can be used within API documentation pages, maintain them as API evolves.

In Conclusion

This research currently looks at some of the common practices I'm seeing across over 20 areas of a modern API life cycle. Unlike other industry research that looks at the API sector, my research is not focused on the venture capital view of the sector, or analysts looking purely from the service provider perspective. I am trying to look at the common patterns when it comes to providing APIs, from the perspective of the platform, its consumers, end-users, as well as the overall industry. My objective is to identify the common patterns necessary for striking the right balance, that makes all of this APi stuff actually work--not just where the $$ is.

This research is constantly changing, and evolving, and this is just my recent attempt to provide a draft snapshot at the essential building blocks across my research. Each area of my research usually involves relevant APIs, and companies who are delivering services and tools in the space, including the common building blocks employed by APIs, which are also the feature sold by service providers. The unique view this particular draft guide provides, is that I am just dumping what I'd consider to be just the essential building blocks of API operations, in hopes of providing an executive overview of my research of the API space--minus the APIs and companies where I extracted the patterns.

Unfortunately my research is pretty robust, and this essential guide is now pushing almost 50 pages (so much for executive guide). While I will work to refine this research, I anticipate the addition of a number of resource links as well, which will only swell its pages. I hope it still will provide a checklist that API providers can use, while planning their API operations. Check back often, as I'm sure there will be regular updates to take advantage of, capturing the latest in what I consider to be the essential building blocks of the API sector.

Thanks for Tuning Into My Research!

Remember -- You Can Find All Of This On APIEvangelist.com, If You Want To Get Involved!

{"logo":"API Evangelist"}

@kinlane

@apievangelist