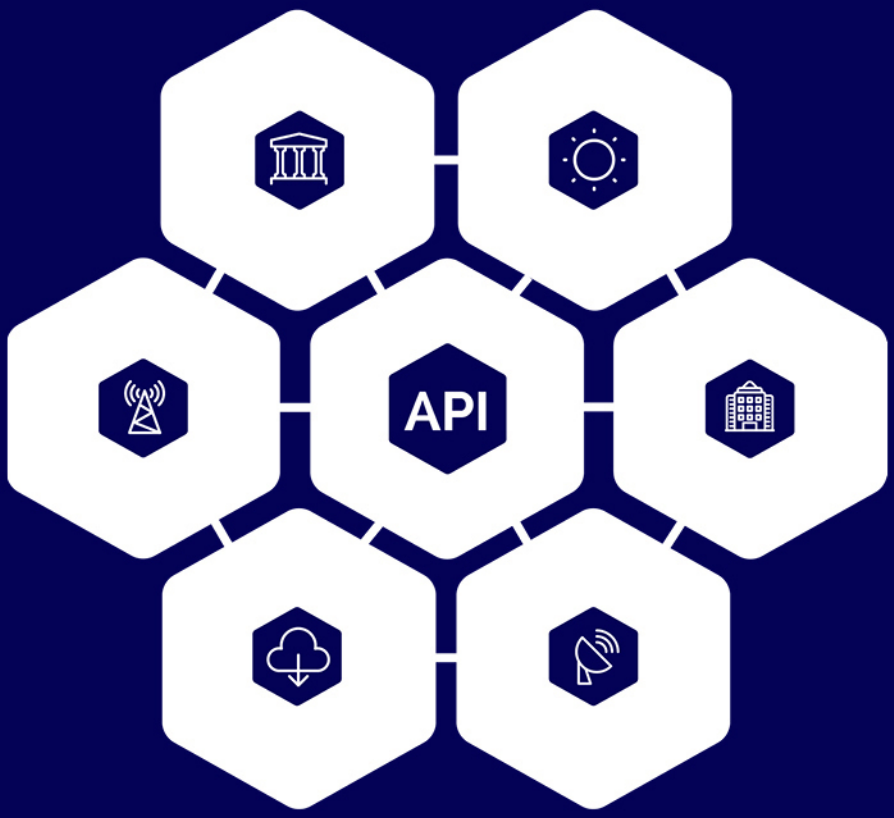
DEVELOPER PORTAL
COMPONENTS



API documentation patterns - part I

Kristof Van Tomme
Kathleen De Roo

PRONOVIX
Developer portals

Developer Portal Components - API documentation patterns part 1

Pronovix

This book is for sale at
http://leanpub.com/developer-portal-components-api-documentation-patterns-part-1

This version was published on 2017-03-08

Tweet This Book!

Please help Pronovix by spreading the word about this book on Twitter!

The suggested tweet for this book is:

Developer Portal Components: API documentation patterns & best practices - free white paper by @pronovix #APIthedocs
https://leanpub.com/developer-portal-components-api-documentation-patterns-part-1

The suggested hashtag for this book is #APIthedocs.

Find out what other people are saying about the book by clicking on this link to search for this hashtag on Twitter:

https://twitter.com/search?q=#APIthedocs

Contents

A note from the authors

Hello!

First of all: thank you for downloading our white paper. We are excited that you are about to start reading **part I of our API documentation patterns white paper series**. We based its content on the blog post series that we published on pronovix.com[1] during the last months of 2016.

We have more news. We are planning two new series about documentation patterns, concerning strategies and technologies. We will start publishing blog posts on a regular basis on pronovix.com soon. After that, we will, similar to our previous series, add a few extras and bundle up the content into *Developer Portal Strategies. API Documentation Patterns Part II* and *Developer Portal Technologies. API Documentation Patterns Part III.*

If you would like to keep informed about the blog posts and white papers to come, then we absolutely recommend you to **subscribe to our Developer portal mailing list**[2]!

If you have any questions, would like to ask for further information or share your thoughts on developer portals, or if you have any feedback on the white paper, you can reach the authors at developerportals@pronovix.com.

Need any help planning, implementing, or optimizing a developer portal? Our company, Pronovix[3] specializes in developer portals. We have a team dedicated to customizing the Apigee (now part of Google Cloud) developer portal for customers around the world. Contact us here[4] for more information!

Enjoy!

The authors.

(Jan 2017)

---

[1] https://pronovix.com/blog/developer-portals-best-practices-documentation-patterns
[2] http://eepurl.com/bF0ztz
[3] https://pronovix.com/
[4] https://pronovix.com/contact-us

1. Developer portals documentation patterns: introduction

While developer portals always target a comparable audience (developers) and need to achieve comparable outcomes, it is remarkable how different most popular portals are. We conducted an in-depth information architecture study of the documentation types used on developer portals **searching for patterns**. This is the first of 3 white papers in which we make an attempt at extracting best practices - all in order to help you build a better developer portal.

In our series we'll give a thorough overview of the documentation components used on developer portals. For a general introduction and for more information about developer experience check out the following blog posts:

- Best API documentation practices[5] (Programmable Web) and advice[6] (Parse)
- A post by Brad Fults about what we need to do to understand and satisfy our audiences[7]
- UX researcher and Pronovix colleague Kata listed UX tips for developer portals[8]

We started our research with browsing through an in-house research on the documentation characteristics of 30 developer portals. We selected 10 sites:

- Twilio[9]
- DigitalOcean[10]
- CenturyLink[11]
- Mashape[12]
- Apigee[13]
- IBM Cloudant[14]
- Keen IO[15]

[5] http://www.programmableweb.com/news/web-api-documentation-best-practices/2010/08/12
[6] http://blog.parse.com/learn/engineering/designing-great-api-docs/
[7] https://bradfults.com/the-best-api-documentation-b9e46400379a#.unshza1fz
[8] https://pronovix.com/blog/best-practices-and-ux-tips-api-documentation
[9] https://www.twilio.com/docs/
[10] https://developers.digitalocean.com/
[11] https://www.ctl.io/developers/
[12] http://docs.mashape.com/
[13] http://apigee.com/about/developers
[14] http://www.ibm.com/analytics/us/en/technology/cloud-data-services/cloudant/
[15] https://keen.io/docs/

- Asana[16]
- Mattermark[17]
- Dwolla[18]

We will focus on documentational categories, labels, subcomponents, information architecture, and structural patterns, not on the content itself. Our purpose is to offer ideas and insights, share what we noticed and learned.

The words *docs* and *documentation* often cover different parts of the developer portal. A few examples:

- *Docs* = quickstart, tutorials, API reference, guides, SDKs (Twilio[19])
- *API docs* on the CenturyLink overview page[20] links to the CenturyLink reference page[21]
- *Documentation* on the Asana overview page[22] links to the section getting started, tutorials and examples[23]

Developer portals in this research

We based our research on the developer portals of 10 companies. To obtain a relevant sample, we included companies that have been praised for the way they structure documentation, companies with community sections for a large and active audience, and companies with a complex API offering that requires a custom portal.
The companies we opted for already appeared in a previous in-house research focussing on URL patterns and documentation characteristics. The results of that research offered us an excellent starting point to evaluate documentation patterns. The following is the list of companies we included, with their USP (in their own words) and a link to their developer portal.

Twilio[24] - a Cloud communications platform for building SMS, Voice & Messaging applications on an API built for global scale.
DigitalOcean[25] - a simple and robust cloud computing platform, designed for developers.
CenturyLink[26] - a reliable local provider of high speed internet, phone and TV services to homes as well as large and small businesses.

[16] https://asana.com/developers
[17] https://support.mattermark.com/
[18] https://developers.dwolla.com/
[19] https://www.twilio.com/docs/
[20] https://www.ctl.io/developers/
[21] https://www.ctl.io/api-docs/v2/
[22] https://asana.com/developers
[23] https://asana.com/developers/documentation/getting-started/overview
[24] https://www.twilio.com/docs/
[25] https://developers.digitalocean.com/
[26] https://www.ctl.io/developers/

Mashape[27] - building leading open-source tools and services for managing Microservices and APIs.
Apigee[28] - The API platform for digital businesses. Manage API complexity and risk in a multi- and hybrid-cloud world with security, visibility, and performance.
IBM Cloudant[29] - Cloudant is the distributed database as a service (DBaaS) built from the ground up to deliver fast-growing application data to the edge.
Keen IO[30] - Keen IO is a set of powerful APIs that allow you to collect, analyze, and visualize events from anything connected to the internet.
Asana[31] - With tasks, projects, conversations and dashboards, Asana enables teams to move work from start to finish.
Mattermark[32] - Search companies and investors to effortlessly create actionable lists of leads.
Dwolla[33] - Dwolla's robust ACH APIs make it easy for businesses of all sizes to integrate a powerful bank transfer infrastructure into their native platforms quickly.

[27] http://docs.mashape.com/
[28] http://apigee.com/about/developers
[29] http://www.ibm.com/analytics/us/en/technology/cloud-data-services/cloudant/
[30] https://keen.io/docs/
[31] https://asana.com/developers
[32] https://support.mattermark.com/
[33] https://developers.dwolla.com/

2. Overview pages

The importance of overview pages

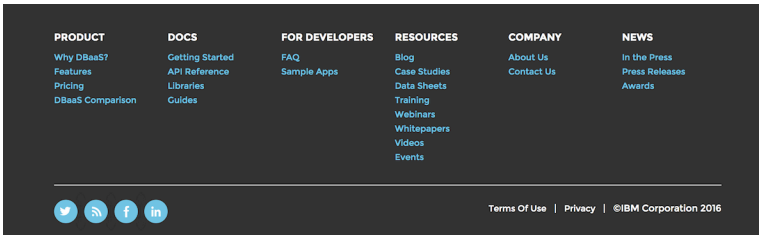
Even though every page is now page one because of search engines like Google, a large group of developers will end up exploring your documentation starting from one of your portal's landing pages. That is why it is really important to provide a front page that gives an overview of the available documentation, an entrance page to the portal where developers can start exploring: the "overview page". Ideally overview pages clearly refer to (all) documentational components (onboarding documentation, API references, guides, tutorials, support resources and SDKs) and provide links to subpages. To help developers process the information, links to subpages will be chunked according to a classification system. The chosen classification system will also typically influence the information architecture of the portal. There are 3 strategies for structuring information. Overview pages can follow:

- **The Documentation type**: it is tempting to group information by type, e.g. all the getting started tutorials, or all the API references, guides,... But often these labels will be ambiguous, as a tutorial can encompass both actionable and inspirational information: e.g. how do I do XYZ - versus - look at all the cool things you can do with our API. The labels for documentation types also require a developer to know under what label they will be able to find a certain type of information.
- **Products**: On a lot of documentation sites information is organised per product. E.g. developer portals often give an overview of all the API products that are documented on the portal. The problem with this approach is that developers will often not know what API product they need. This is even worse when APIs have non-obvious product names.
- **Functional grouping**: A better way to organise information is to start from the developer's perspective. If you have several APIs on your portal, are there broad categories of objectives, or functions that you can group your APIs under? Alternatively can you help developers identify what area they should go to based on their goals?

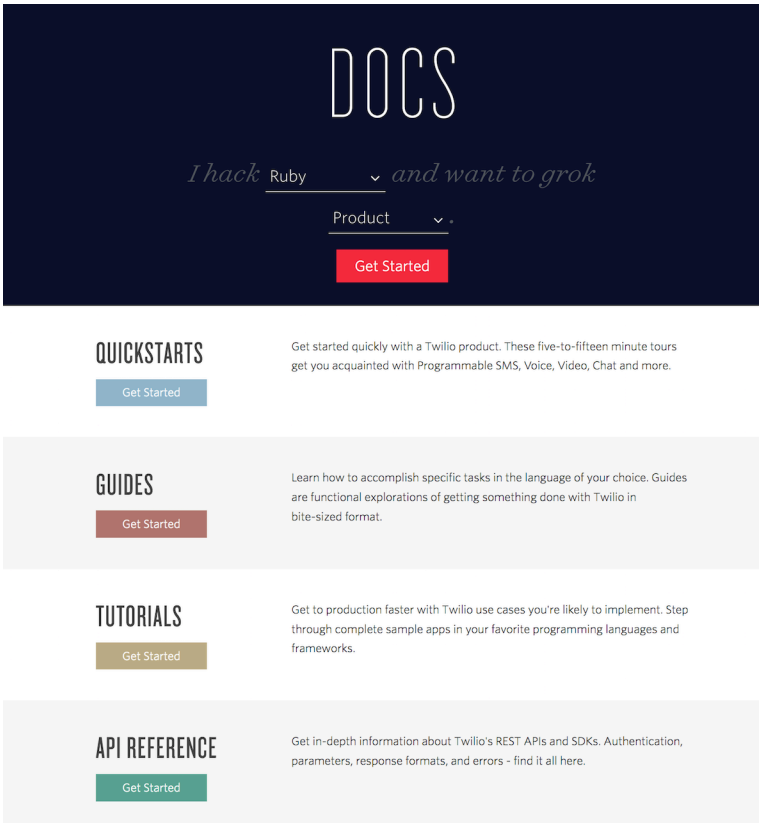
Information structuring is an important part of information architecture. UX researchers have a number of tools like Stakeholder interviews, User Research and Analysis, Persona creation and Card sorting that can help explore what grouping would be most intuitive for your stakeholders.

Structural components

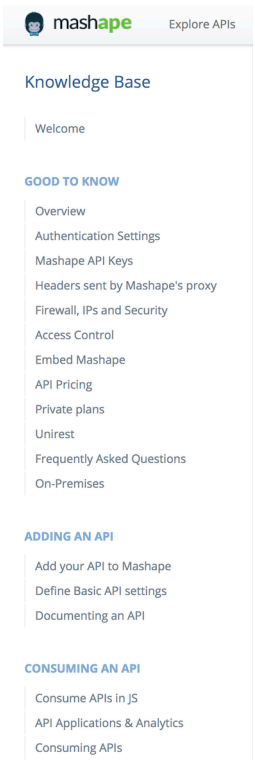
In our research sample, pages could be divided in the following areas: the body section, sidebar, header and footer:



Example of components represented in footer section (IBM Cloudant former "for developers" page, screenshot Oct 2016)



Example of components represented in body section (Twilio)



Example of components represented in sidebar menu (Mashape)

Sidebar menus (like the Mashape overview page[34]) were rare - more often the components appeared in header/footer and/or body section, but even then the reviewed sites sorted the documentation components differently. Some examples: Twilio[35] and Asana[36] make a clear distinction between the documentation concepts that are represented in the body section and the ones that are mentioned in the footer, while Keen IO[37] includes several (sometimes identical) terms in header, footer and body section that provide links to the same subpages.

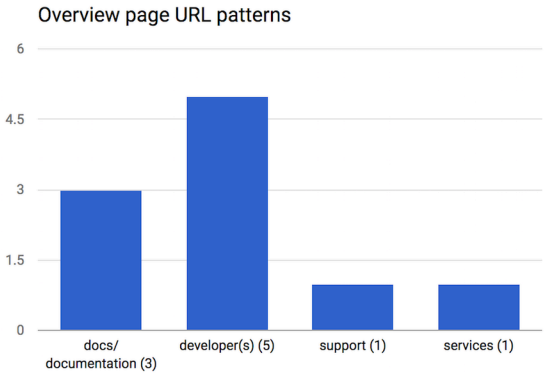
Mental model, what is the function of your developer portal?

In our sample of 10 developer portals we checked if we could find a connection between URLs and the language used to label the overview page. The overview page URLs differed: 3 contained “docs” or “documentation”, 5 “developers”, 1 “services” and 1 “support”.

***Note**: This is an updated version of our original blog post on pronovix.com[38], in which we wrote that the portals of IBM Cloudant and Apigee both had two overview pages: one for developers, one for*

[34] http://docs.mashape.com/
[35] https://www.twilio.com/docs/
[36] https://asana.com/developers
[37] https://keen.io/docs/
[38] https://pronovix.com/blog/developer-portal-components-part-1-overview-pages

docs. The overview pages of the other companies also often referred to separate "docs" pages, but then solely listed reference documentation. IBM Cloudant's and Apigee's "docs" pages, however, also contained links to other documentation components (like tutorials or guides), so we defined them as "docs overview pages". We've come to believe, however, that we should treat the "docs overview pages" of these two companies as subpages. IBM Cloudant itself led us to this decision: the company recently changed its overview page named "for developers" into an overview page with a more clear categorization of links to the docs page and other documentation resources[39]*. Subsequently, we've also decided to handle Apigee's developer overview page (with URL "developers") as the main overview page, as it provides a link to the docs (references) page as well.*



Though Dwolla[40]'s URL includes "developers", the browser page title (next to the favicon) is "Dwolla API Documentation". Often we see that docs/documentation in the URL is represented in the main title or headline chosen for the documentation part on the overview page; URLs with "developer" mainly had marketing focussed titles:

URL SNIPPET		MAIN TITLE / HEADLINE FOR API DOCUMENTATION ON OVERVIEW PAGE
Docs / documentation (2 sites)	→	Docs / documentation
Docs / documentation (1 site)	→	Developer resources
Support (1 site)	→	How can we help you today?
Services (1 site)	→	(Company name)
Developer (5 sites)	→	Build a platform for teamwork
	→	Simple and powerful API for ACH transfers with no per transaction fees
	→	Tools and Support
	→	Power your apps with our simple API
	→	Developer Tools to Build and Manage Awesome APIs

Our small-scale research **indicates that there might be a correlation between the use of "docs" or "documentation" in the URL and "documentation" as headline or the most important word to**

[39] http://www.ibm.com/analytics/us/en/technology/cloud-data-services/cloudant/

[40] https://developers.dwolla.com/

name the overview page. You could understand this as an indicator of a difference in how developer portal owners think about the function of their portal:

1. **Portal as a documentation hub**: a developer portal is the documentation, a meta-object that describes a product, the canonical source of all product information.
2. **Portal as a support system**: a developer portal is there for troubleshooting, to help resolve support requests, a form of self-service system.
3. **Portal as a developer resource**: the portal is there to help developers use APIs to fulfil their needs.

We believe that it is best to use the 3rd mental model, in which the experience is framed in the context of the developer's experience. Identifying different developer goals and structuring the site to help them achieve those goals.

Covering all the necessary documentation

The overview pages contained a range of documentation types (some portals had blog posts, try it out sections, use cases or suggested eBooks, …). There is no one-size-fits-all answer for what to include, but the following are a few tips to make your overview page useful and appealing:

- Show what you have in store - developers should get an overview of the whole portal.
- Make sure that the most important words of your overview page are reflected in the URL and in the headline/main title. In other words: use clear terminology that reflects what's listed on the overview page.
- Provide easily recognizable concepts (this is connected to visual design elements).
- Make a distinction between body sections and header/footer. Don't hide main documentation components in headers and/or footers. Try not to include everything in the body section, provide a place for the details in the header/footer. (We prefered Twilio's approach with main components in the body sections, while the footer provides links to other documentation topics.)

Make it as easy as possible for developers to find the way to the documentation they might need: guide them.

3. Onboarding pages

The onboarding page, a crucial component for developer portals

Developer experience on first contact is arguably the single most important job of a developer portal. In some cases your API might only have a few minutes to hook a developer, if it isn't immediately clear where developers can start their integration efforts, they might leave and never come back. This is why onboarding pages are so important. In this article we'll list the various roles onboarding pages need to fulfill. We'll also explore how the developer portals we reviewed expose them and how they are labeled.

Different onboarding for different audiences

Depending on the objective and the prior experience of a user, an onboarding page needs to fulfill 3 different roles:

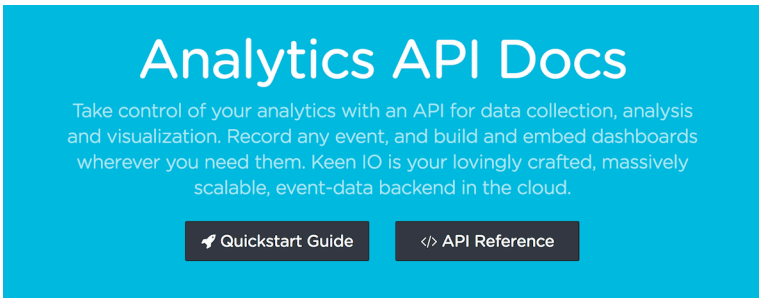
- **Evaluate**: Help developers and managers decide if your API will have all the necessary features for the implementation they are planning.
- **Educate**: You should never assume that a developer will have a full understanding of the industry. A developer might not be familiar with some of the domain language (professional vocabulary used in an industry) you are familiar with (e.g. "dunning" in recurring payments). But developers will, at least initially, not know what they don't know, so they will not start reading your thesaurus. In his Write the Docs talk[41], Michael Meng reported on the findings from his research with Andreas Schubert and Stephanie Steinhardt. In their research they found that the inclusion of conceptual documentation in onboarding materials might help speed up developers.
- **Accelerate**: The most obvious function of onboarding materials is to give developers a clear list of all the steps that need to be taken to make a complete and proper integration. Code samples in onboarding materials can also help developers avoid common pitfalls.

[41] https://www.youtube.com/watch?v=soQSOBwiXdA&ab_channel=WritetheDocs

Onboarding in the developer portal's information architecture

Ideally, the overview page (e.g. frontpage or API product landing page) should provide a link to the onboarding page. In our research 10 (out of 10 developer portals) had the onboarding documentation as a main component on their overview pages.

- DigitalOcean[42] labeled "get started" as a subcomponent of "guides".
- Dwolla[43] interpreted "Get Started" as a "Set up Sandbox guide".
- Apigee's docs page[44] hid the onboarding documentation in the sidebar menu grouped under their respective product sections.



Example of a Quickstart guide component on Keen IO's overview page (Keen IO)

How to call your onboarding section

"Get started" is a widely used term. We noticed two customs:

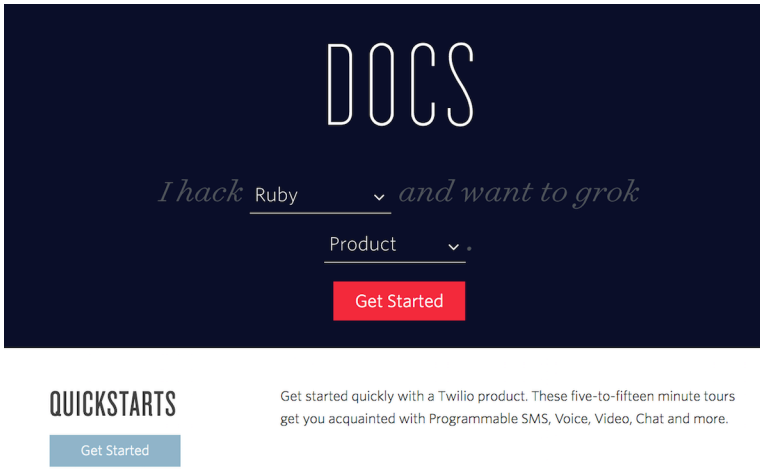
- *Get started = The how-to-start guide.* The developer portals we researched used quickstarts, guide, get started, basics, getting started and walkthroughs mostly to refer to a beginner's guide or introductory article(s) and rarely to describe in-depth articles.
- *Get started = The call-to-action (CTA) button* (refers to subpage).

Onboarding subcomponents

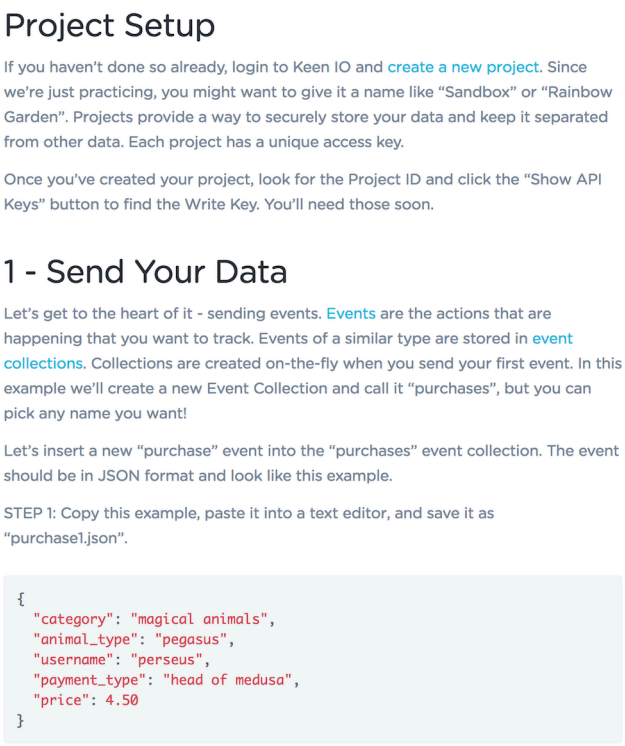
The formats of the onboarding documentation differed. We found:

[42] https://developers.digitalocean.com/
[43] https://developers.dwolla.com/
[44] http://docs.apigee.com/

- *Questions and answers* that explain basic facts about the API;
- *Videos* that engage your users visually;
- *Introductory and/or in-depth articles* describing the company, API and/or used terminology;
- *In Real Life examples*;
- *Code snippets* to help developers get acquainted with your API product.



Example of "Get started" as a CTA button and "Quickstarts" as a how-to-start guide - note how the text also starts with "get started" (Twilio overview page)



Project Setup

If you haven't done so already, login to Keen IO and create a new project. Since we're just practicing, you might want to give it a name like "Sandbox" or "Rainbow Garden". Projects provide a way to securely store your data and keep it separated from other data. Each project has a unique access key.

Once you've created your project, look for the Project ID and click the "Show API Keys" button to find the Write Key. You'll need those soon.

1 - Send Your Data

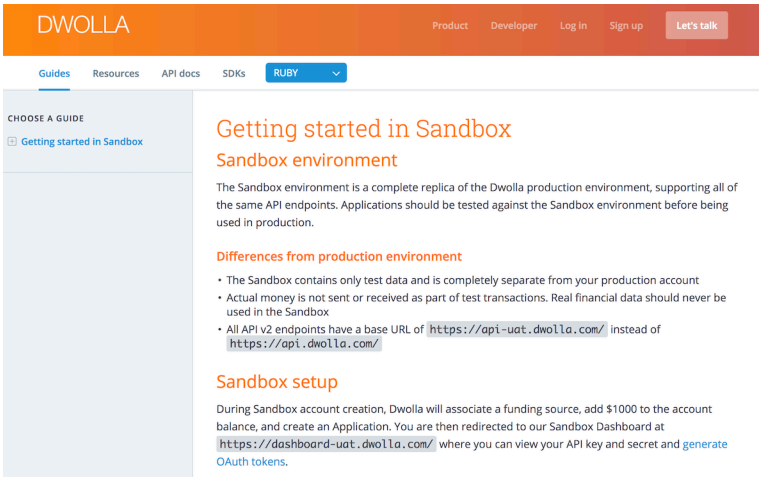
Let's get to the heart of it - sending events. Events are the actions that are happening that you want to track. Events of a similar type are stored in event collections. Collections are created on-the-fly when you send your first event. In this example we'll create a new Event Collection and call it "purchases", but you can pick any name you want!

Let's insert a new "purchase" event into the "purchases" event collection. The event should be in JSON format and look like this example.

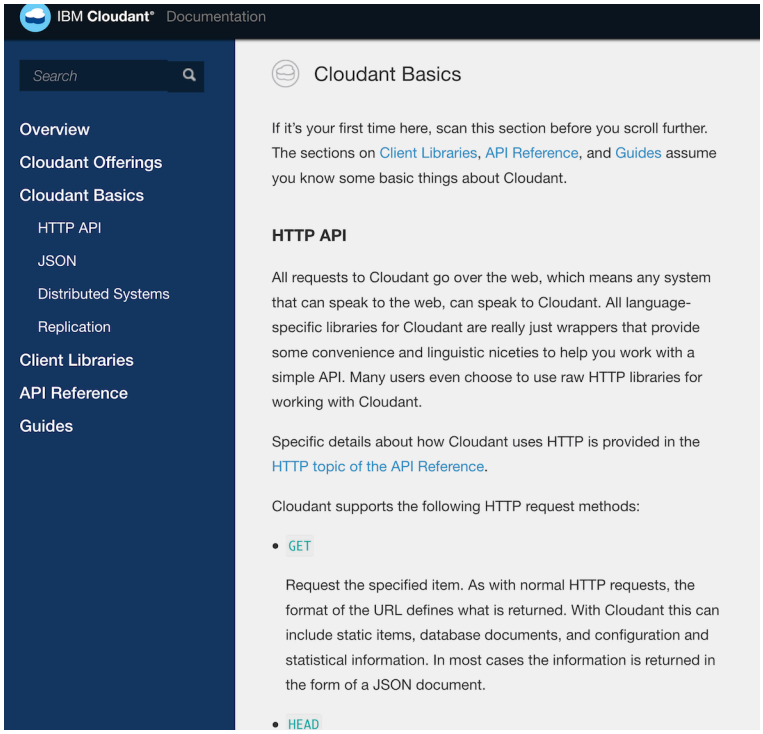
STEP 1: Copy this example, paste it into a text editor, and save it as "purchase1.json".

```
{
  "category": "magical animals",
  "animal_type": "pegasus",
  "username": "perseus",
  "payment_type": "head of medusa",
  "price": 4.50
}
```

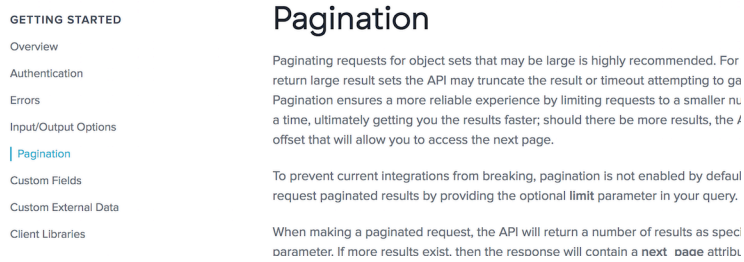
Example of a Quickstart guide with code snippets (Keen IO)



Example of a Getting started with code snippets (Dwolla)



Example of clearly indicated basic information about specific Cloudant terminology ("Basics", IBM Cloudant)



Example of Getting started used as a general term for introductory ("Overview") and in-depth articles (with or without code) (Asana)

Onboarding best practices

Onboarding documentation is an essential part of a developer portal. Main tips:

- Treat onboarding documentation as a main component on your overview page.
- Keep in mind what the onboarding documentation should cover (introductory information, in-depth documentation)
- choose your terminology accordingly. Use obvious titles: developers should get the meaning immediately.

- CTA text alternatives for *Getting started* could be *Learn more* or *View*.
- Differentiate between an introductory article about the product and a real how-to-start guide with code snippets by using:

  - *Getting started guide*, *how-to-start guide*, *quickstart guide* or *onboarding guide* solely for explaining core features (including code snippets) and
  - *Introduction* or *Basics* for general information in a nutshell (eg. about specific words used for the API of a certain company), in plain English. We do not recommend the term Overview for short introductions, in order not to confuse it with the overview page of the developer portal.
- Allow immediate API testing through code samples or provide a sandbox environment.
- Test your assumptions about labeling and validate your user experience. Read up on UX and DX research to do this yourself, work with a contractor to avoid familiarity bias, or contact us[45] - we are setting up a procedure and a team specifically to evaluate developer experience on developer portals.

Note: in this post, we focussed on onboarding documentation. We found, however, that specific site builders also treated tutorials as "getting started" topics or the other way around, others put onboarding documentation in the category of "Guides" on the overview page. More on that in next chapter.

The research we referred to from Professor Meng's group still needs to be published, but you can watch his talk at Write the Docs Prague on YouTube[46].

For further information on improving API documentation terminology: check API Documentation with meaningful names by Jan Christian Krause[47] *and his talk at Write the Docs Prague*[48].

[45] https://pronovix.com/contact-us#
[46] https://www.youtube.com/watch?v=soQSOBwiXdA&ab_channel=WritetheDocs
[47] https://github.com/jankrause/wtd-2016
[48] https://www.youtube.com/watch?v=YOBVWfmRHzc

4. Guides and Tutorials

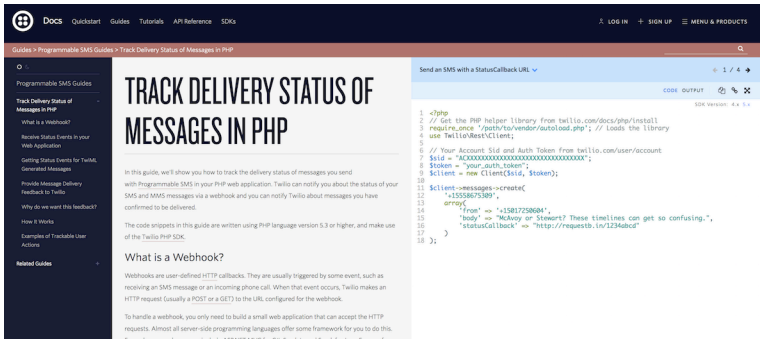
Guides and tutorials: documentation for differently skilled developers

If the primary function of a developer portal's onboarding page is to help developers self-select their knowledge level and to create an expectation of learning, then the function of the guides and tutorial pages that are linked from an onboarding page is to fulfill this expectation: to design an experience that will be as close as possible to the learning requirements of key audience segments. In this post, we will explore how the sites we reviewed included guides and tutorials in their information architecture, we'll look at what they covered, and finish with listing ideas to maximize their effectiveness.

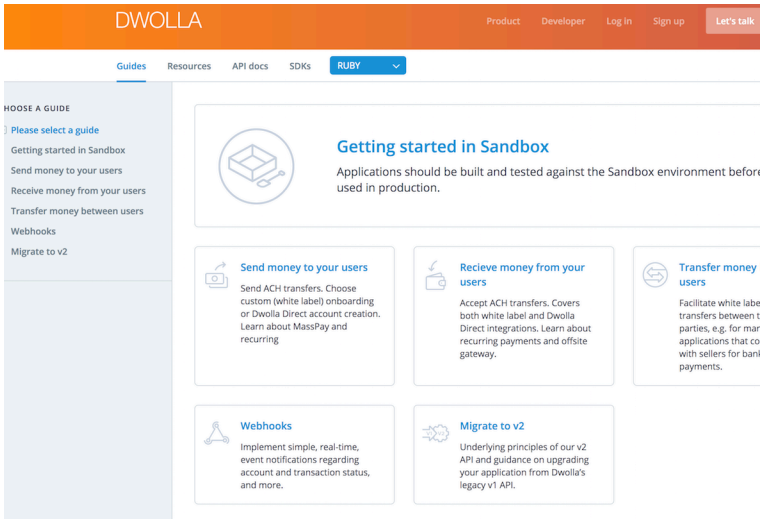
Sharing theoretical and practical knowledge

Guides and tutorials are not always perceived as distinct words, that is why there was some overlap in their usage on developer portals. In our developer portal component series we want to make the distinction between these two words to be able to address two separate needs. Here, we will use guides to focus on theoretical aspects and tutorials to focus on practical aspects of API related topics:

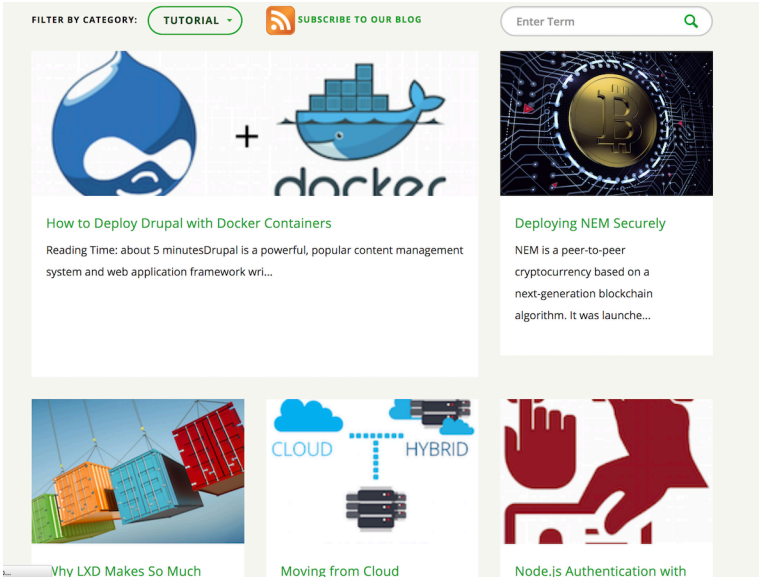
- **Guides = teaching material**: Guides cover a range of topics, set up in plain English, they help your audience to gain knowledge about your product and reference documentation. Guides often explore typical problems and use cases a developer might experience using your product (or refer to more detailed examples elsewhere - e.g. SDKs).
- **Tutorials = how-tos**: Tutorials provide step-by-step explanations, that get developers to experience your product as fast as possible. Ideally tutorials are clear how-to's that are interactive and focus on a specific part of your product's functionalities.



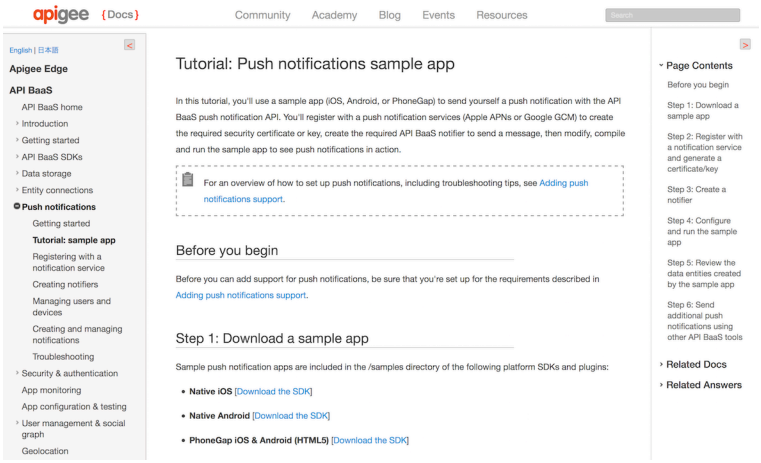
Example of a guide with code snippets (Twilio)



Example of a Guides page (Dwolla)



Example of a Tutorials page (CenturyLink)



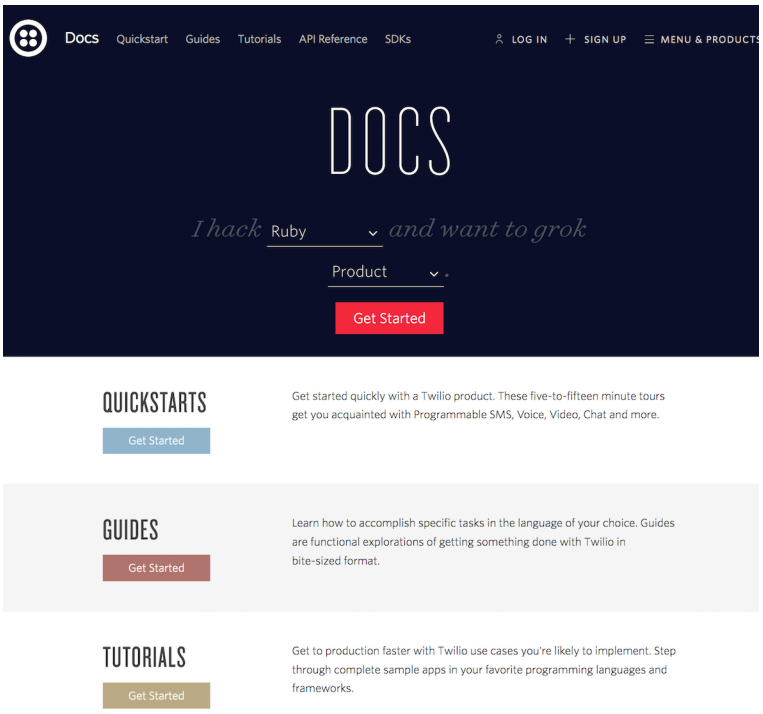
Example of a tutorial with step-by-step explanations (Apigee)

Guides and tutorials in the information architecture of the developer portal

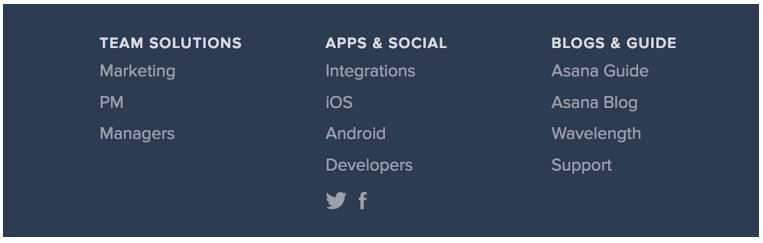
A mismatch between your audiences and the information on your developer portal can be disastrous for your product's adoption. Developers can be particularly impatient, if they can't immediately find the information they need on your site they will use a search engine to look for the answers they need. At Signal, a developer conference, Twilio presented the reasoning behind the recent redesign of their developer portal[49]. They found that they had much better results when they focussed their

[49] https://www.twilio.com/signal/schedule/6Gjpnzch0sggKeey0yigko/new-product-session-3

tutorials on code samples with as little prose as possible. This is - at least on the surface - in conflict with the findings from Michael Meng's research group[50], that developers with insufficient knowledge of your API's name space need conceptual documentation to be included in tutorials to be able to make the fastest possible progress. So it appears that it might be crucial to target different audiences with different types of information. This is something we want to investigate in future research. The findability of guides and tutorials is crucial and, therefore, we believe that its representation on the landing page (overview page) of the developer portal is important. While most sites in our research tended to treat Guides and Tutorials separately, they rarely added both on their overview pages.

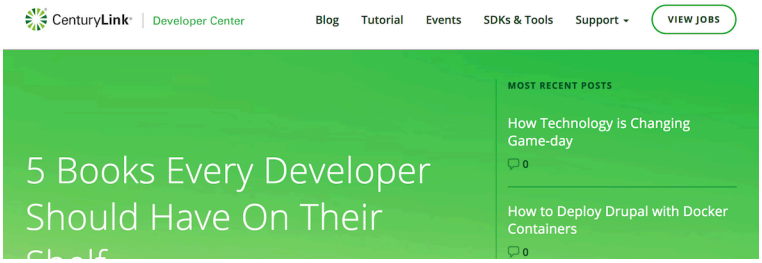


Example of an overview page with a Guides section and a Tutorials section (Twilio)



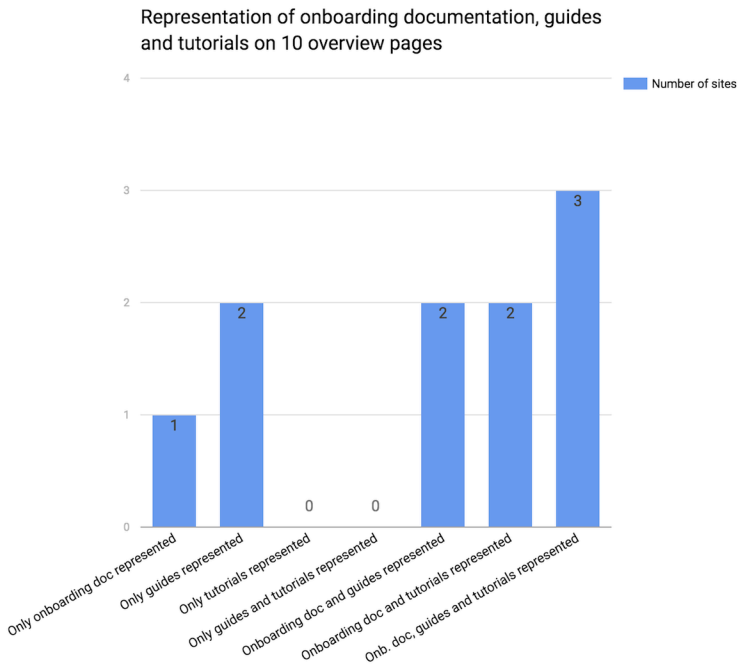
Example of a Guides section in the footer of the overview page (Asana)

[50] https://www.youtube.com/watch?v=soQSOBwiXdA&index=17&list=PLZAeFn6dfHpnN8fXXHwPtPY33aLGGhYLJ

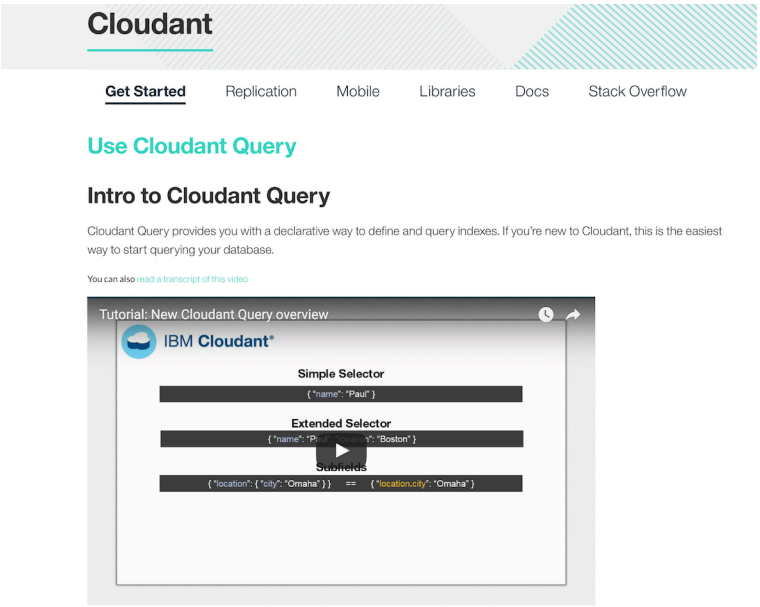


Example of a Tutorials section in the header of the overview page (CenturyLink)

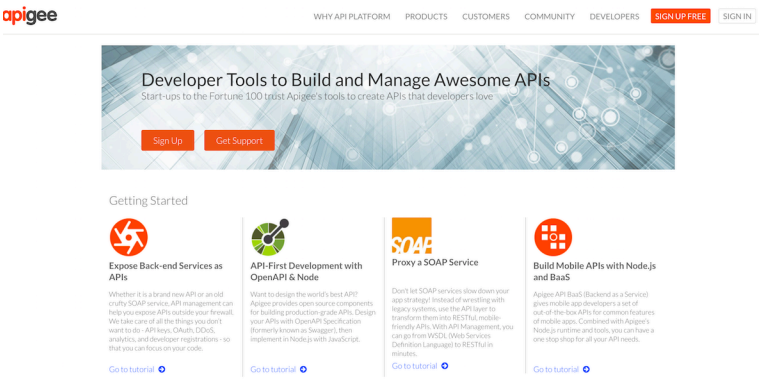
We found a correlation between the presence of different types of labels. The presence of onboarding documentation, guides and tutorials influenced the presence of one another on the overview pages. (note: for the chart we solely looked at the words “guides” and “tutorials” and “get(ting) started”/”quickstart (guide)”/”basics” for onboarding documentation). In most cases developer portals with onboarding documentation also had either Tutorials or Guides. This means that the content of Guides and Tutorials is (at least partly) perceived as being similar and not complementary.



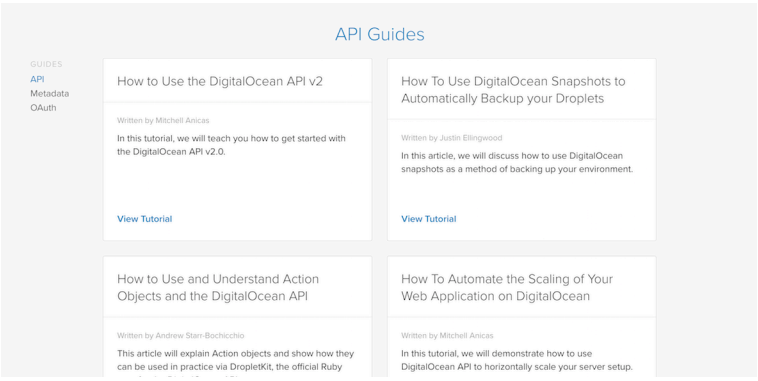
We also found examples of guide-tutorial and getting started-tutorial merging - on overview pages and subpages:



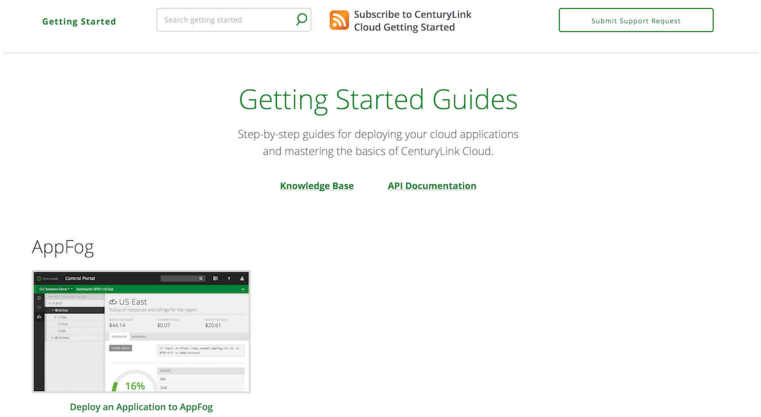
Example of a video tutorial as Getting started documentation (IBM Cloudant)



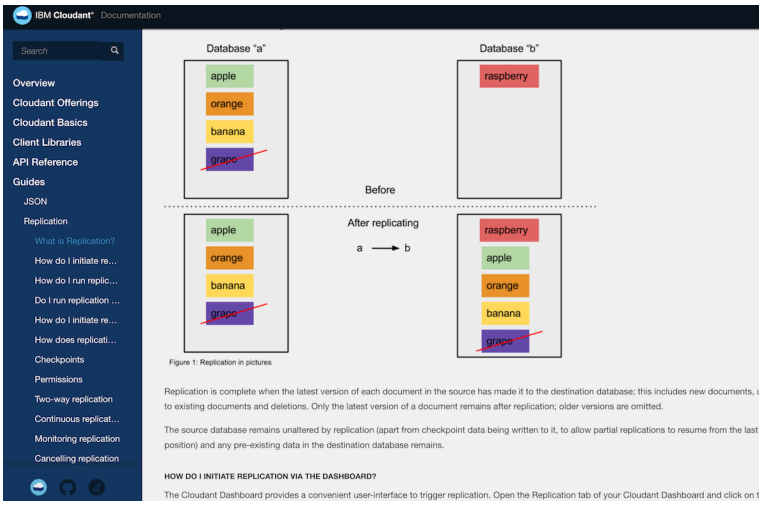
Example of an overview page with tutorials listed as "Getting Started" topics (Apigee developers overview page)



Example of Tutorials as a subcategory of a Guides page (DigitalOcean)



Example of Getting started guides, indicated as step-by-step guides with video tutorials (CenturyLink)



Example of a guide with theoretical explanations. Note that "How do I..." topics are categorized into the "Guides" section in the sidebar menu. (IBM Cloudant docs page)

Guides are not tutorials, tutorials are not guides. Their purpose, however, is similar: they help developers to learn about an API product. Our research samples showed that most developer portals use both guides and tutorials. To pursue clarity and increase findability, we recommend treating guides and tutorials as main components on the overview page, either listed separately, or grouped together (as "Guides and Tutorials").

How to call the components

If you use both guides and tutorials it might not be immediately obvious for your users what you mean with these labels and what the difference is between them. This is probably why Twilio[51] has added an audience description next to the headers for the different sections of their documentation,

[51] https://www.twilio.com/docs/

instead of marketing copy about their products that would have been better for SEO purposes. Labeling guides and tutorials for this post was not evident, as developer portals often have different ideas about what to put under which umbrella, e.g.:

- "Examples and Tutorials" labelled as one category (Asana)
- The expression "How-to" was once used for referring to a specific guide (Asana), once for tutorials (DigitalOcean)

While looking for definitions that would describe the difference between guides and tutorials, we did not find conclusive solutions. We decided to base our definitions partly on James Yu's (Parse) descriptions[52] and partly on general definitions that we found on Wikipedia (a tutorial[53] seeks to teach by example and supply the information to complete a certain task) and in the Cambridge Dictionary (guide[54]: a book that gives you the most important information about a particular subject). The chart that we included above shows that guides and tutorials are popular (guides were present on 7 developer portal overview pages, Tutorials on 5), but not often categorized both as separate components on the overview page (only 3 out of 10). But what about other pages of the developer portal? Just one of the developer portals we reviewed categorized tutorials not only into the guides section, but also on its community page among topics like questions, projects, meetups. If tutorials and guides are not added to the overview page, they are rarely present on subpages of the developer portal.

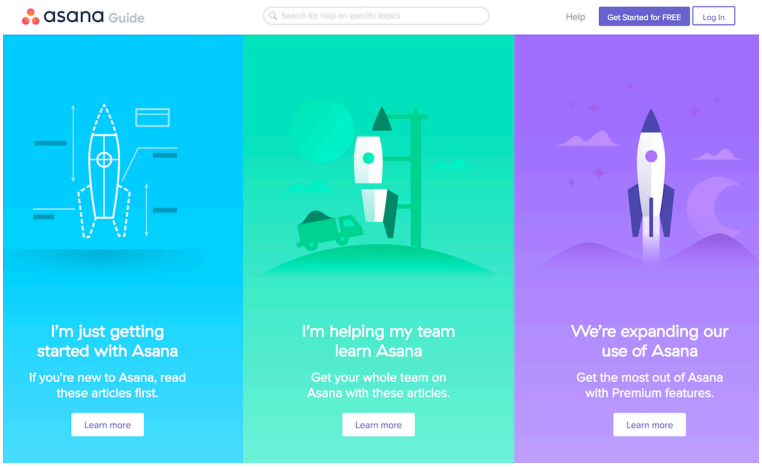
Formats of guides and tutorials

To make guides and tutorial sections more appealing, consider adding:

- interactive elements, like videos, images, CTAs "start the tutorial", arrows
- code snippets
- icons, columns, bullet points, colors to increase the visual experience
- tags to improve topics searchability
- examples of problems, use cases
- links to similar topics and/or discussions or to StackOverflow and GitHub
- categorization elements, e.g. based on code language
  You could also turn your tutorials into an interactive sequence of easily processable steps. Definitely check out Twilio's presentation about the things they did to improve conversion rates on their tutorials[55]. Our colleague Steve analyzed the findings of this talk in a blog post[56]. The guides and tutorials sections we studied were mostly categorized into topics. Asana

[52] http://blog.parse.com/learn/engineering/designing-great-api-docs/
[53] https://en.wikipedia.org/wiki/Tutorial
[54] http://dictionary.cambridge.org/dictionary/english/guide
[55] https://www.youtube.com/watch?v=hTMuAPaKMI4&ab_channel=TwilioAPI
[56] https://pronovix.com/blog/5-things-learn-twilios-documentation-overhaul

guide[57], however, provided a guides overview page that is not based on topics, but on level of advancement and purpose:



Asana Guide: not topic, but purpose focussed

Best practices: point out the difference between theory and practice

Developers start/continue at different levels. Make sure they can quickly notice the difference between onboarding documentation, guides and tutorials. A few tips:

- Make sure the difference is clear (through visual design, usage of unambiguous terminology: e.g. add a description of what you will list like Twilio[58] did)
- Make your tutorials interactive: e.g. include videos, bullet points, code examples.
- The onboarding guide should be clearly reflected on the overview page and treated as a main element: don't hide it among the guides.

[57] https://asana.com/guide
[58] https://www.twilio.com/docs/

5. Reference pages

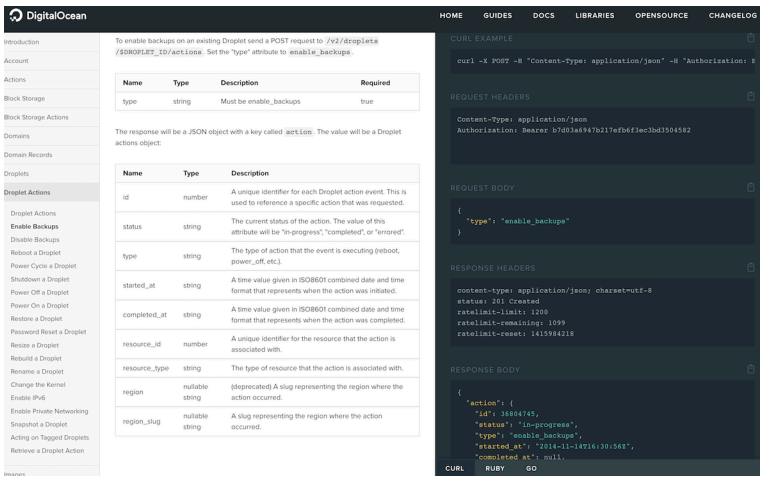
Reference pages: an API's inventory and checklist

Once a developer knows how to use your API, they will need detailed instructions on how to build the actual integration. Experienced developers already familiar with an API, including its creators, will have a hard time completing an integration without access to the API reference. That is why developers will often refer to them as "the API documentation". This is a dangerous practice, as it implies that an API reference would be the only documentation an API needs, we talked about this in our previous posts.

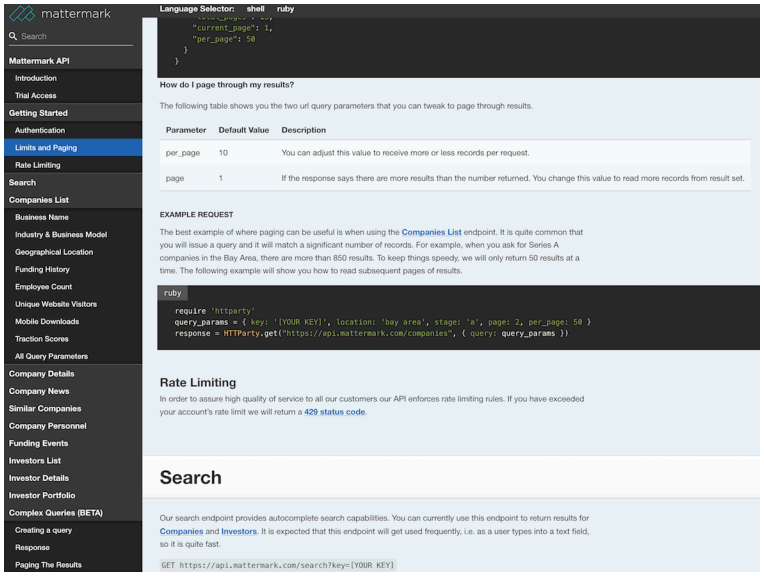
In this post we'll show how the developer portals in our research sample implemented their reference pages, compare their formats and labels, and try to deduct best practices you could apply on your developer portal.

Structure and subcomponents of the reference page

Developers will use the API reference during development, so it is very important to include elements that minimize the time it takes to find a specific function or parameter and to provide tools that help them browse and understand your endpoints. There is much less variation between reference page implementations than between the documentation types: developer portals usually list the reference topics on the left, and categorize descriptions and code snippets into one or two columns on the right.



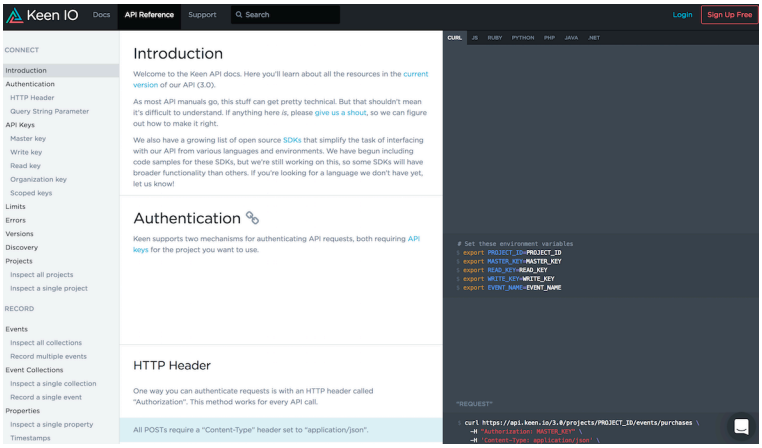
Example of an API reference page with descriptions and code snippets in separate columns (DigitalOcean)



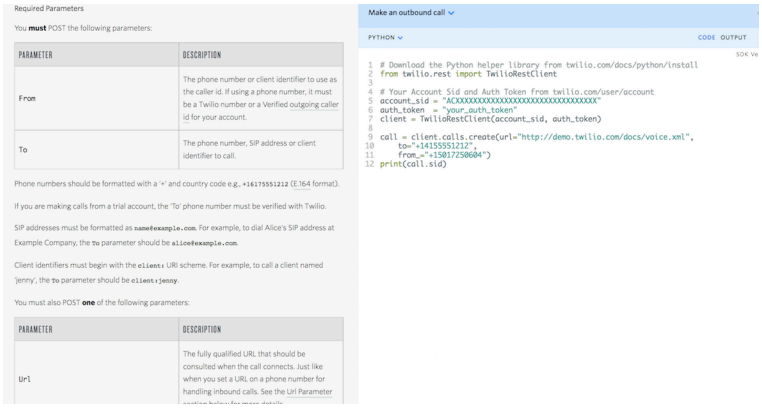
Example of an API reference page with descriptions and code snippets in one column (Mattermark)

In our research sample, API reference pages optionally contained the following subcomponents:

- Code snippets / sample code to provide examples of how developers can use and test the API code.
- Code language selectors so developers could switch the code samples to their preferred implementation language.
- Descriptions of terms and processes that a developer needs to know to understand an API.
- Search bars to speed up developers.
- CTAs (Call to Action), e.g. to code snippets on GitHub, community support on StackOverflow, a registration form, or to contact the support team.
- Collapsible columns to unhide/hide content: developers can get more details when they need them, but they are not shown by default so developers don't get overloaded. This also makes it easier to scan the docs for relevant information.



Example of an API reference page with CTAs, search bar and code language selector (Keen IO)



Example of descriptions in the API reference (Twilio)

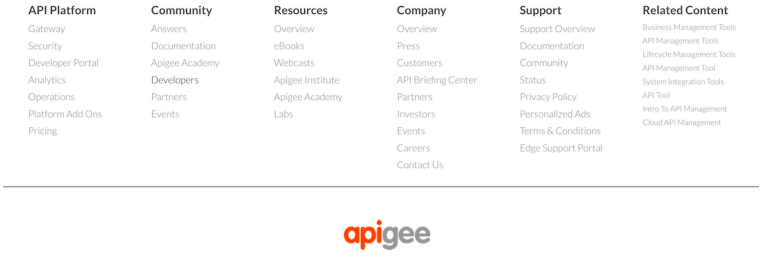
Reference pages in the developer portals' information architecture

In our research we reviewed the overview pages of 10 companies.

- 9 overview pages had a direct link to topics, descriptions and reference code.
- The "Documentation" component on DigitalOcean's overview page linked to a subpage with 3 CTAs (API, OAuth, Metadata). CTA "API" links to the reference docs.

We observed 2 patterns:

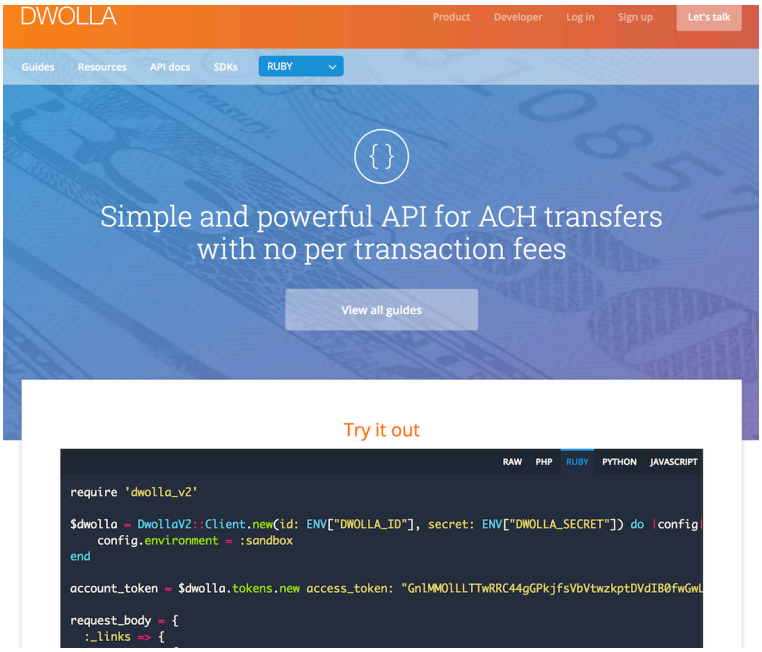
- a direct link to the reference documentation
- a second overview page that lists additional documentation topics (e.g. metadata, support).



Example of "Documentation" component linking to the docs page (Apigee overview page)

In a sample of ten, three developer portals (Mashape[59], IBM Cloudant[60] and Apigee[61]) listed the API reference together with other documentation components (like onboarding documentation, guides and tutorials) in a hierarchical sidebar menu. To get to the reference docs developers often need to unfold this menu first. The other 7 developer overview pages provided a link to a subpage dedicated solely to the API reference.

Dwolla took developer engagement even further, with a sample code in the "Try it out" section on their overview page that enables developers to test and explore code without even visiting the reference documentation.



Example of sample code (in "Try it out" section) on the overview page (Dwolla)

[59] http://docs.mashape.com/
[60] https://docs.cloudant.com/
[61] http://docs.apigee.com/

Various names for API reference pages

The terminology used to refer to API reference pages was:

- API reference (mentioned 3 times)
- Documentation (3)
- API docs (2)
- (Company name) API (2)
- API documentation (1)
- Documenting an API (1)
- Reference (1)
- API (1)

Best practices to set up a reference page

Reference pages should be user-friendly:

- Add concepts to your API reference where possible to help developers understand the reference code (we already mentioned Michael Meng's talk at WTD Prague 2016[62], who did research on how developers read documentation).
- Include links to other site sections on the developer portal (eg. FAQs, guides, ...) or to external community resources like GitHub and Stackoverflow.
- Make sure you include code snippets in various languages.
- Clear terminology with an easily findable link on the overview page helps your developers to find the code they need.
- Include color identifiers, arrows e.g. to indicate chapters on the reference page.
- Add descriptions of processes and definitions of domain language (industry specific terms) used in the code.
- Make the sidebar (if any) simple: provide an understandable table of content or faceted search system.

For further reading on best practices, read Kata's post with UX tips for API documentation[63].

APIs are difficult to describe when you're not a developer. Have you already read Laura's blog post What is an API[64] to help you out?

[62] https://www.youtube.com/watch?v=soQSOBwiXdA&ab_channel=WritetheDocs
[63] https://pronovix.com/blog/best-practices-and-ux-tips-api-documentation
[64] https://pronovix.com/blog/what-api

6. FAQs, Forums and other Support Resources

Self-service support is arguably the most important role of a developer portal. Without proper documentation, API teams will spend countless hours on introduction workshops and other training and support efforts. In this post we'll analyze the characteristics of a number of support resources and look at how they take self-service even further and involve users to develop information about the problem areas in an API's use. We'll list pros and cons for the different resource types, look at their place in the site architecture, and finally propose best practices.

Support resources are a documentation format

Even the best documented APIs, that complement their API reference with plenty of onboarding tutorials, also have a wide range of support solutions on their developer portals. Twilio, often praised for the quality of their API documentation, includes a list with support articles[65] on its portal. The better the portal, the more support solutions available. On first sight this might seem counterintuitive: on portals with great documentation you would expect less support. This does make sense however if you see support artifacts as a type of API documentation. When writing the docs, it is hard to provide materials that will fulfill all requirements to everybody: sometimes you might overlook a demographic, or find out that some people use different words or mental models to understand your service. When you capture the support interactions with these people, they can become a valuable documentation resource that explains problematic concepts in the customer's words, improving the documentation and reducing the number of people that need staffed support.

Support types: peer-to-peer support and staffed support

Staffed support

Questions are answered by a support team, and where possible reused as documentation content. Examples:

1. FAQ pages
2. Knowledge Base pages

[65] https://support.twilio.com/hc/en-us

3. Support pages (email, contact form, live chat)

Peer-to-peer support

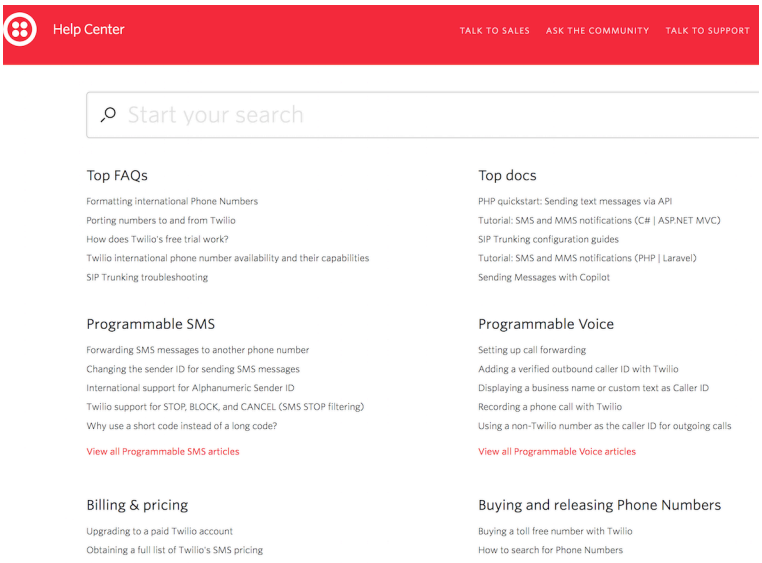
Interactions between developers are facilitated and result in community support. Staff might occasionally answer questions, or lend authority to an answer, but most questions are answered by customers or partners. Examples:

1. Community sections / Forums
2. Third party community pages, like GitHub and StackOverflow

1. FAQ pages

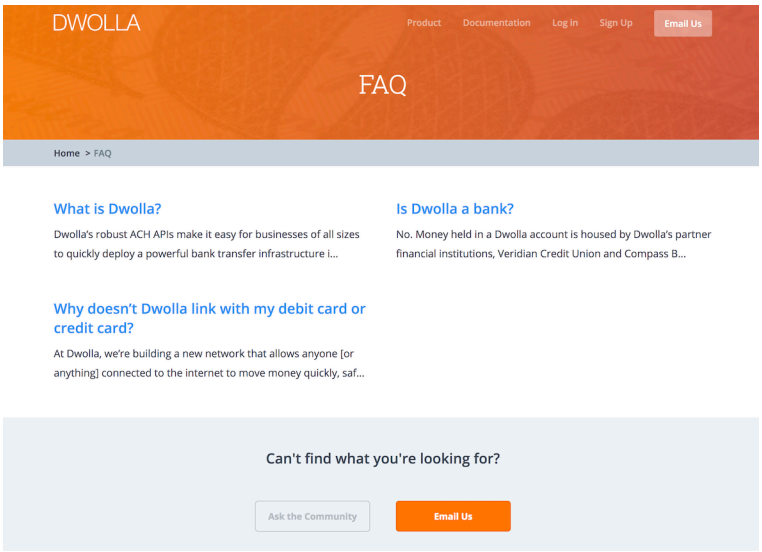
"Frequently Asked Questions" or also "Questions" list issues that have been previously addressed in a question-answer format. FAQs are popular help sources, because they are easy to produce. They are however delayed(issues appear online after a answer-evaluate-publish workflow) and incomplete: a lot of FAQ pages have sentences like "Can't find what you're looking for?" and CTAs (call-to-action buttons) to ask for support on not yet listed problems. Developers don't read, they scan pages and should be able to grasp the content of the FAQ page right away. Some examples of how that could be done:

- Structure FAQs into categories. Longer lists (with more than ten topics) are easier to scan when they are divided into categories.



Example of an FAQ page with articles listed in various categories (Twilio)

- List FAQs chronologically or without clear order. This works when the FAQs only include a small number of topics (not more than 5). When used in this way FAQs function primarily as marketing material that helps overcome common objections.



Example of an FAQ page with a list of 3 questions, without clear order (Dwolla)

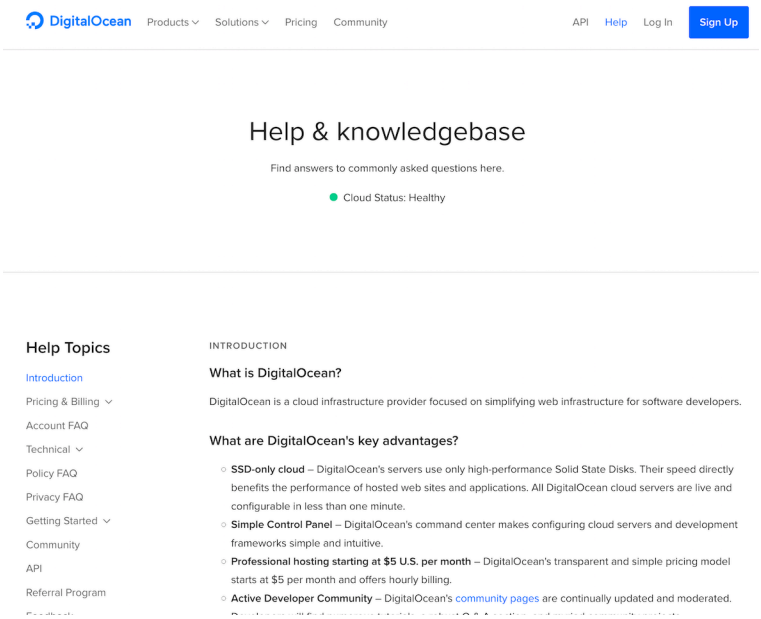
- Add a search bar, and provide a scannable result list.

In our research sample, 5 portals out of 10 provided an FAQ page.

2. Knowledge Base: the upgraded FAQ page

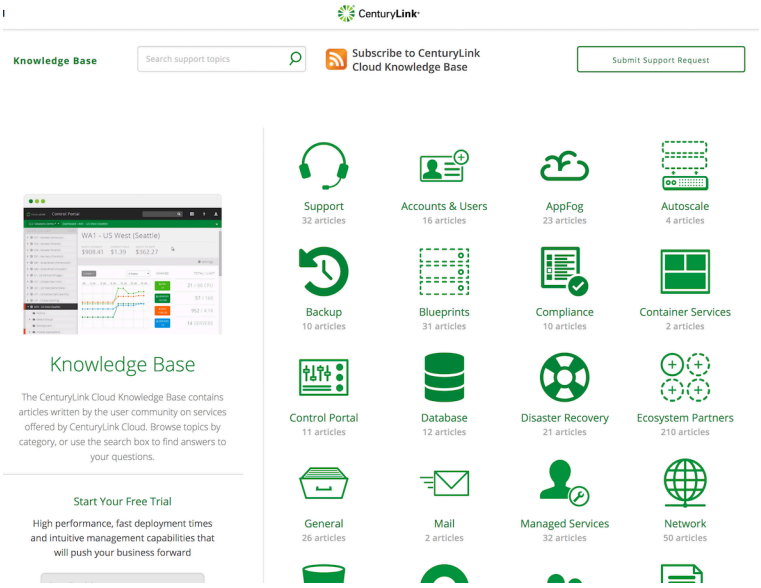
Knowledge Base pages list issues and answered questions, often wrapped in articles. The articles are structured into categories. Developer portals added the following elements to improve the content scannability:

- A table of content



Example of a Help and Knowledge base page: the topics are represented in the sidebar menu table of content (DigitalOcean)

- Visual design elements, like icons

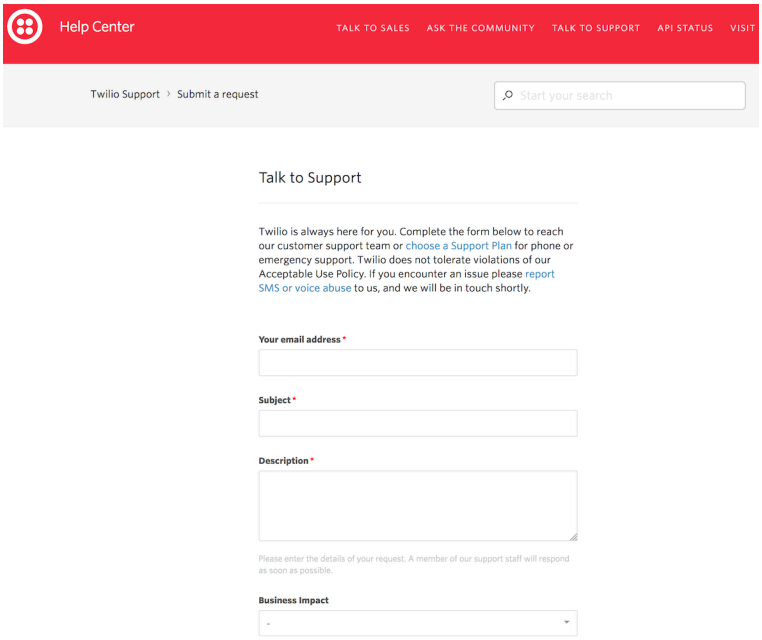


Example of Knowledge Base categories with icons. The company also added a page description: "The CenturyLink Cloud Knowledge Base contains articles written by the user community on services offered by CenturyLink Cloud. Browse topics by category, or use the search box to find answers to your questions." (CenturyLink)

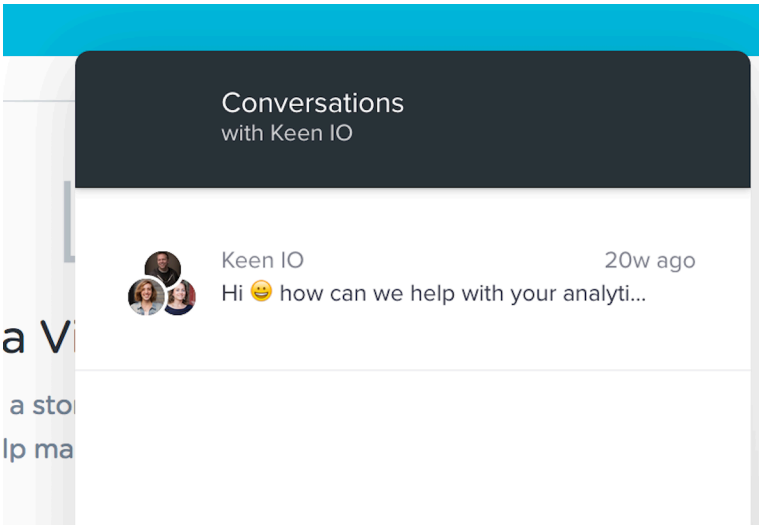
In our research sample, 4 portals out of 10 provided a Knowledge Base page.

3. Support pages

Users are invited to send an email, chat live or submit a detailed contact form to the company's support team.



Example of a contact form (Twilio)

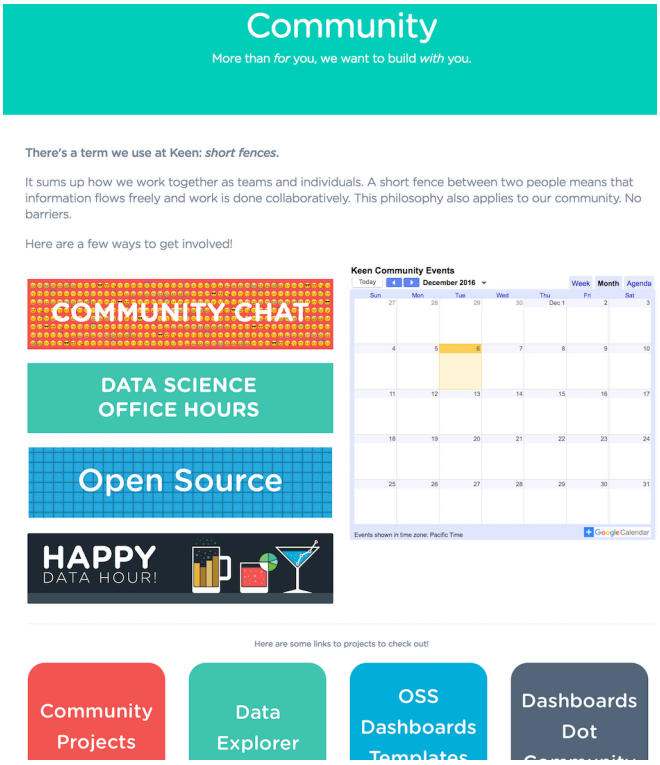


Example of a live chat window (Keen IO)

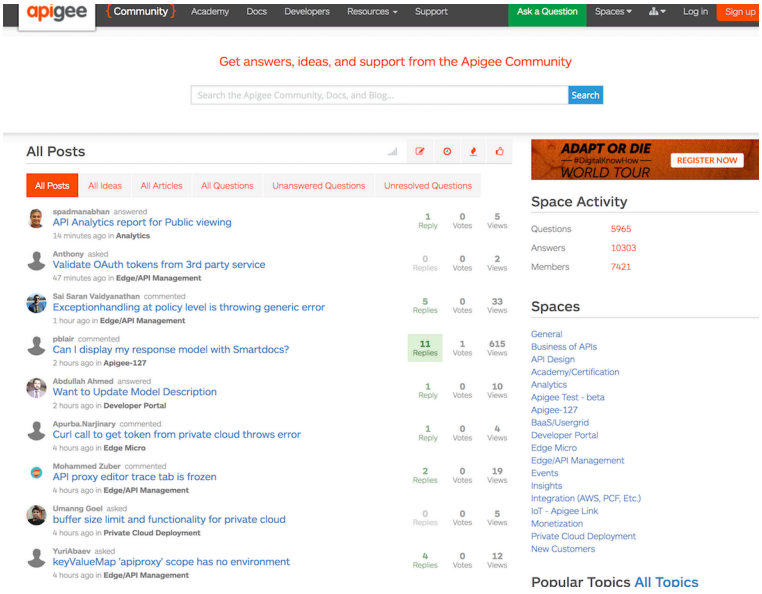
Every developer portal in our research provided a way to contact their support team.

4. Community section: problem solving among peers

We made the distinction between community portal pages and community sections. A community portal page typically contains links to a number of support pages. A community section involves users in peer-to-peer support interactions on-site. Its page structure is very similar to that of GitHub repositories and StackOverflow pages, but search options, filtering options and terminology might differ. Most on-site community sections will also lack the sophisticated gamification and intrinsic reward systems that make third-party community solutions so successful.

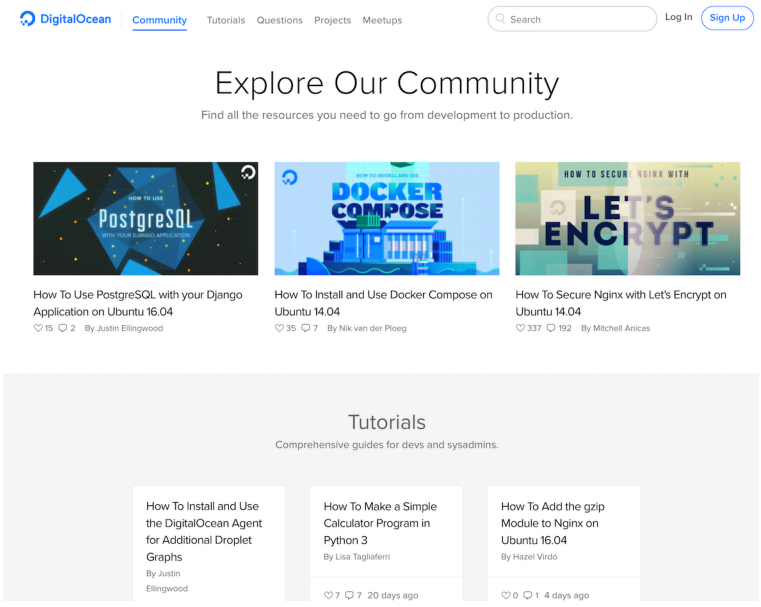


Example of a Community portal page with an agenda of future meetings, a link to the company public slack channel, and a list of projects (Keen IO)



Example of a Community section with a table of content (topics), a search bar, and filtering options (Apigee)

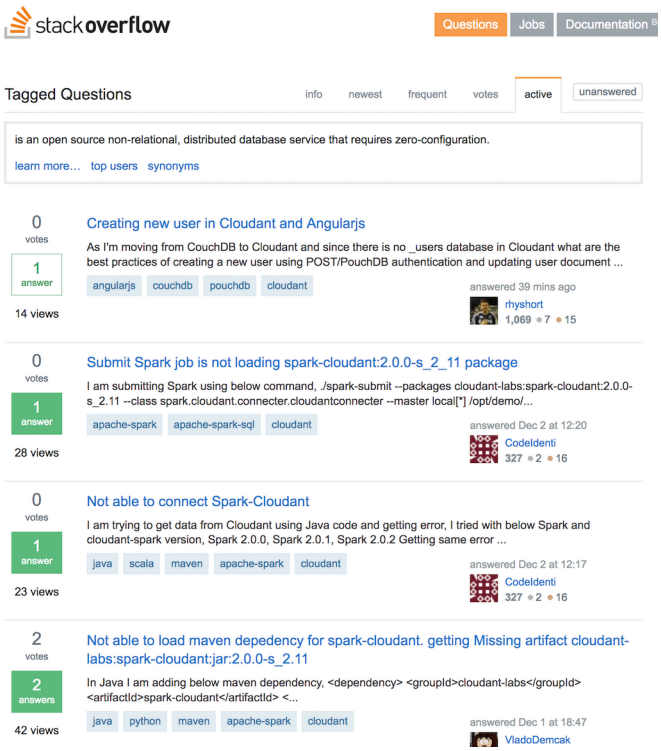
In our research, 3 portals (Keen IO, Dwolla, DigitalOcean) had a community portal page and 2 portals (Apigee and Dwolla) a community section. DigitalOcean's community portal page mainly contained links to tutorials.



Example of community portal page with tutorials (DigitalOcean)

5. Third-party community platforms, like GitHub and StackOverflow

7 developer portals provided third-party community platforms. We found links to Twitter[66], Slack[67], Facebook[68] and Google Groups[69], but GitHub[70] and StackOverflow[71] were by far the most popular. The popularity of third party community platforms is a double-edged sword, they have a lot of content, but that also means that users might need to invest more time to explore the content. Public, third-party community platforms have a number of other less ambiguous advantages: A lot of potential contributors will already have an account and might have built a reputation on the platform that can be used to moderate their contributions. Developers typically also already have an intuition for the culture, rules, and interface of third-party community platforms.



Example of the Cloudant StackOverflow page (IBM Cloudant)

[66] https://twitter.com/?lang=en
[67] https://slack.com/
[68] https://www.facebook.com/
[69] https://groups.google.com/forum/#!overview
[70] https://github.com/
[71] http://stackoverflow.com/

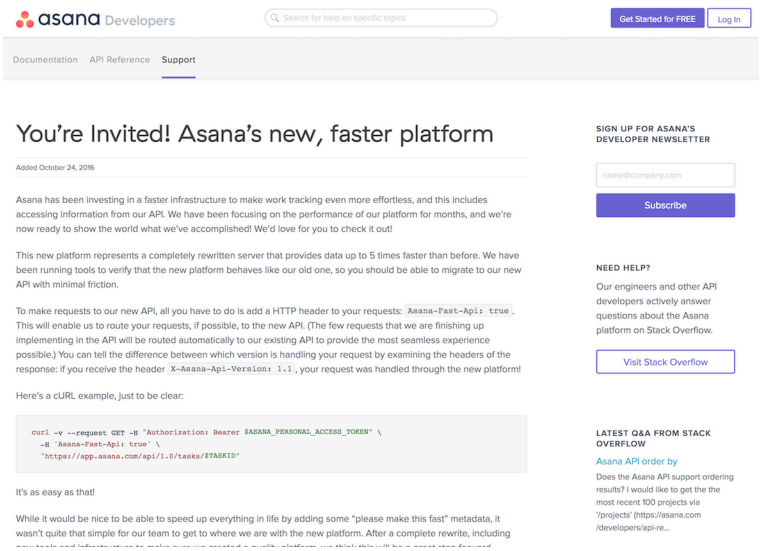
Choosing support types: support strategies

The 10 companies in our research combined between 2 and 4 support resources. We deducted the following strategies:

- All developer portals provided at least one option to contact a support team.
- FAQ and Knowledge Base pages rarely occurred together. DigitalOcean provided both, but used confusing browser page titles (“Questions and answers” for the FAQs and “Frequently asked questions” for its “Help and Knowledge base”).
- On-site community sections and third-party community platforms had a similar function:
- 4 out of 10 portals that didn’t have an on-site community section had set up a GitHub and/or StackOverflow project instead.
- Twilio used an “Ask the Community” CTA to link to their StackOverflow page, Dwolla to link to their community section.

Through their support pages, the developer portals exposed information about UX and content architecture. We observed two patterns:

- Scanning and scrolling takes too much time. The articles are not collapsible, a table of content is missing, the used support terminology is confusing.
- Scanning and scrolling takes a minimum of time. Users are able to switch easily between information sources, search functionality works efficiently, community members can interact.



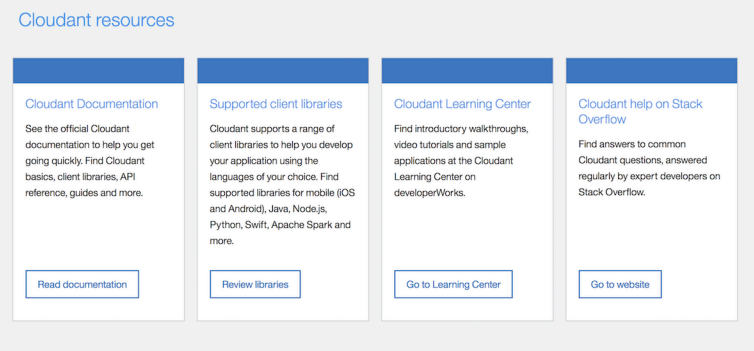
Example of a support page with a list of articles without a table of content, a link to StackOverflow and a possibility to contact the support team (Asana)

Setting up a support strategy helps to choose the right support systems for your developer portal. An understanding of the following aspects is essential:

- Audience
- Budget
- Maturity of your community
  Providing users with peer-to-peer support, like forums or on-site community sections, might not be the best choice for young API products. Building a community of developers takes time and an empty forum can undermine trust. Forums also need a control mechanism to handle spam and unwanted comments. When you are not 100% sure if you will be able to successfully launch a developer community, it is better to invest in staffed support instead that later can evolve into a peer-to-peer support service.

Information architecture of support pages

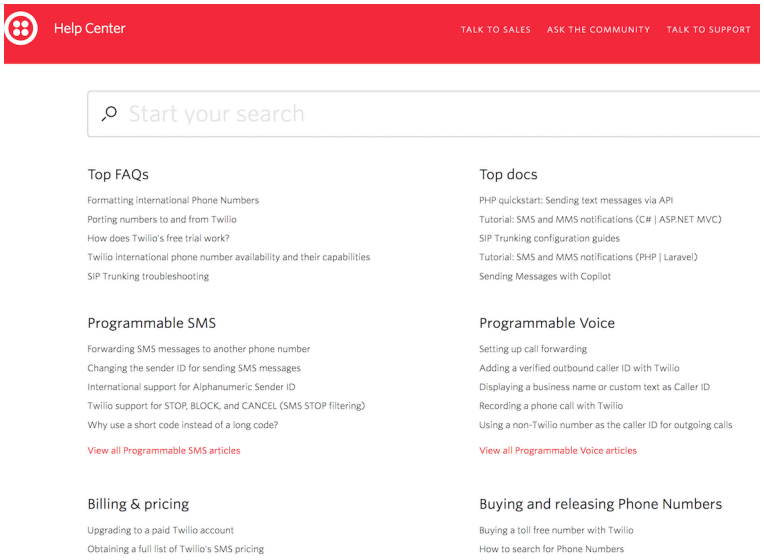
Most of the overview pages (the front pages of developer portals) contain links to FAQ, Knowledge Base and community sections in their header, body and/or footer. CTAs linking to GitHub and StackOverflow pages were regularly implemented on subpages, like support pages or docs pages. Three developer portals also included a CTA to StackOverflow on their overview page. Options to contact the support team appeared on both the overview page and the support subpages.



Example of a CTA to StackOverflow on an overview page (IBM Cloudant)

DigitalOcean[72] categorized its support pages on the marketing site and not on its developer portal[73]. The Mattermark developer portal[74] is the "Help Center" of the Mattermark marketing[75] page.

[72] https://www.digitalocean.com/
[73] https://developers.digitalocean.com/
[74] https://support.mattermark.com/
[75] https://mattermark.com/

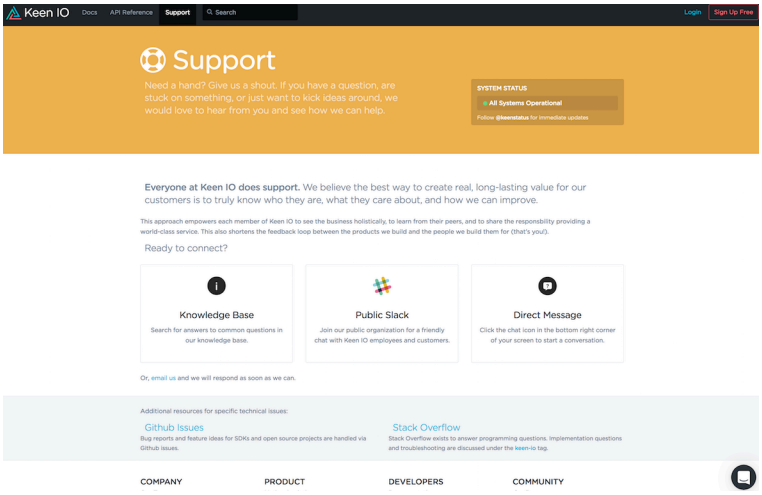


Help Center subpage: Ask the community - Talk to support in header; FAQ in body, categorized by topic (Twilio)

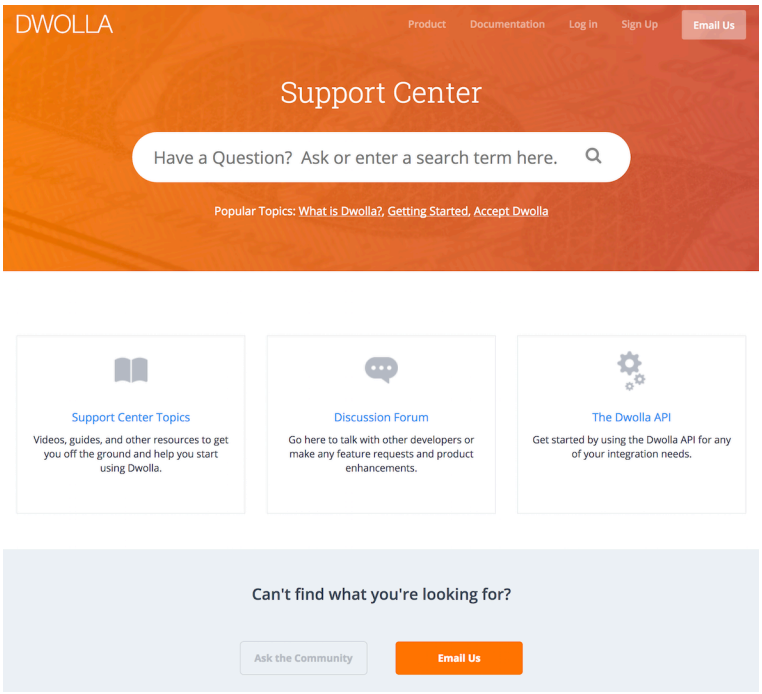
All support options on a single page: the support overview page

A support overview page might help to avoid haphazardly placed links on the portal that could confuse developers. In our research, almost half of the reviewed developer portals provided one. We found it is important to:

- Use unambiguous terminology that immediately shows what the support title covers.
- CTAs - on the support subpages that link back to the support overview page. This might help developers to navigate through the information.



Example of a support overview page with clear terminology, with a link to contact the support team (via email and live support), find answers in a Knowledge Base, on StackOverflow, and GitHub. A link in the footer redirects the user to the community portal page. (Keen IO)



Example of a support overview page with confusing terminology: "Support Center Topics" contains FAQs, an onboarding guide, guides and tutorials. "Discussion Forum" and CTA "Ask the Community" both link to the company's community section and "The Dwolla API" links to the developer portal's overview page. CTA "Email us" allows users to contact the support team. (Dwolla)

Support labels

There is not enough consensus in the labelling of support resources to draw conclusions about best practices, so we've listed all the labels we found here, and will leave their prioritization to a follow up research. Apart from "FAQ", "Frequently Asked Questions", "Knowledge Base", Community, "GitHub" and "StackOverflow" we also found:

- Support
- Support overview
- Help center
- Get Help
- Help
- Account (with subcategory support)
- Answers
- Stuck or need help?

on the overview pages, and

- Support centre topics (linking to FAQ a.o.)
- Discussion forum (linking to the on-site community section)
- Questions
- Help and Knowledge base
- Need help?

elsewhere on the portals.

CTAs to on-site community sections:

- Ask the Community
  CTAs to GitHub and/or StackOverflow:
- Visit StackOverflow
- GitHub issues
- Ask the Community

CTAs to contact the company's support team:

- Talk to support
- Submit a request
- Stuck or need help?
- Get support

- Email us
- Can’t find what you’re looking for? Contact support
- Let us know
- Get in touch
- Direct messaging

Ambiguous names:

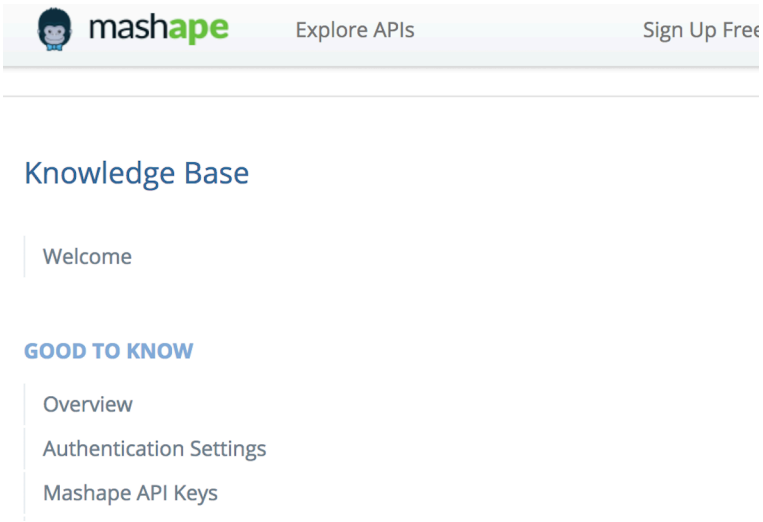
- “Resources” referred to Keen IO’s community portal page; Apigee categorized reports, stories, videos, customer stories and webcasts into “resources”.
- “Ask the Community”: a CTA to Dwolla’s community section and to Twilio’s StackOverflow page.

Support page subcomponents

We regularly found:

- Search bars
- CTAs with links to support pages or to contact the support team
  Subcomponents that improved UX:
- Table of content (e.g. for Knowledge Base categories in the sidebar)
- Live chat options (via icons to open a chat box or via automatic pop-up windows)
- Icons
- Code snippets in articles
- A category with “Popular questions” on the overview page
- CTAs to browse the API reference
- A definition of “Knowledge Base”
  “Knowledge Base” was also the main title of the Mashape[76] docs page sidebar, including API references, tutorials and FAQs.

[76] http://docs.mashape.com/



Example of "Knowledge Base" as the main title of the docs overview page sidebar (Mashape)

Best practices

Ideally, the support team will collaborate with technical writers to create and maintain documentation and support pages. It is crucial to keep them up-to-date and user-friendly:

- An FAQ or Knowledge Base page can be an advantage, but is not a must. Keep the skimming audience in mind: provide a structure that enables developers to scroll through the content quickly, add the total number of articles per category, insert selection criteria (like top articles, latest articles, all questions, unanswered questions) and reference information (like votes) on community pages.
- Provide an on-site community section where developers can communicate with peers. Make the content easily findable. A company GitHub or StackOverflow account could replace a community section on the portal itself.
- Organize FAQ pages in such a way that it makes it easy to explore and extract results from them.
- Provide means to contact the support team.
- A support overview page could help developers to speed up.
- Make support available on the developer portal, not only on the company's marketing page.
- Make sure the terminology is unambiguous (e.g. a tutorial page is not a community section).
- Combine different support resources.

7. Software Development Kits (SDKs)

The main purpose of Platform Software Development Kits and Helper/Client Libraries (we'll use "SDKs" to address these collectively in our writing) is to accelerate and simplify development. A well maintained SDK is a trust signal that indicates the level of support and usage of your API for a language, framework, or development platform. So indirectly SDKs work as social proof[77], that indicates how many communities are already using your API.

In this post, we'll look at how the developer portals in our research sample included SDKs. We'll examine their functions, describe where we found them in the site architecture and deduct best practices.

We'll discuss what kind of SDKs the Portals in our sample used. We'll analyze their choices and evaluate them against the principles that **Taylor Barnett** from Keen IO shared at APIstrat earlier this year[78]. We'll also talk about the strategic choices that need to be made when deciding what kind of SDKs an API should have.

SDKs are part of the API product

SDKs are software development tools that make it easier to build applications. For web APIs that means SDKs are typically API connector libraries that developers can include into their code. Because SDKs implement APIs in language/platform native functions, they can save developers a lot of time. For this reason **developers will often look for an SDK in their favorite language/framework before they even start exploring your API**.

That is why SDKs need to be done right: it is great if you can offer an SDK for a developer's favorite language, but if you offer one, you need to make sure it works. SDKs should be up-to-date, fully tested and well documented. It is inconvenient if an SDK is missing, but it is way worse if you set an expectation and then break it with a buggy SDK.

In our research, 9 (out of 10) developer portals provided SDKs (Twilio[79], DigitalOcean[80], Dwolla[81], CenturyLink[82], Keen IO[83], IBM Cloudant[84], Apigee[85], Asana[86], Mashape[87]).

[77] https://en.wikipedia.org/wiki/Social_proof
[78] http://www.slideshare.net/taylorsoitgoes/creating-a-great-developer-experience-through-sdks"
[79] https://www.twilio.com/docs/libraries
[80] https://developers.digitalocean.com/libraries/
[81] https://developers.dwolla.com/pages/sdks.html
[82] https://www.ctl.io/developers/sdks-tools
[83] https://keen.io/docs/sdks/
[84] https://docs.cloudant.com/libraries.html#supported-client-libraries
[85] http://docs.apigee.com/api-baas/content/sdks
[86] https://asana.com/developers/documentation/getting-started/client-libraries
[87] http://docs.mashape.com/unirest

Why do you need SDKs?

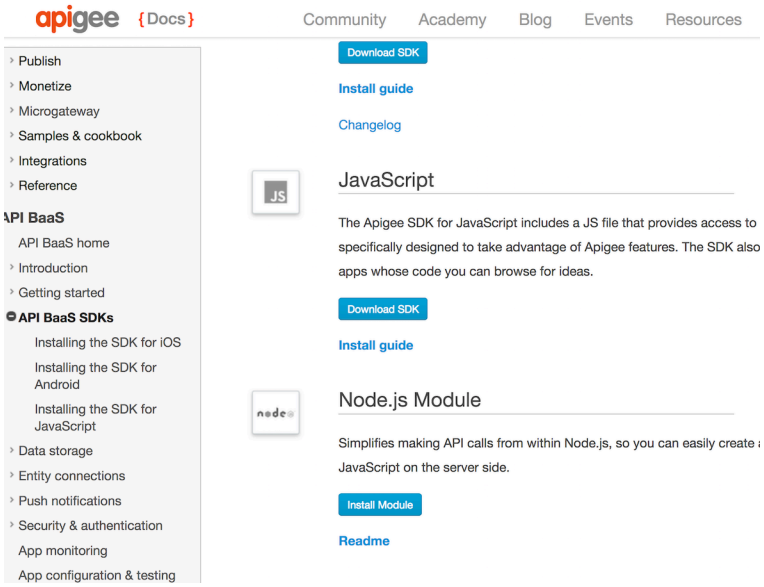
In her 2016 APIstrat talk, Taylor Barnett[88] explained why Keen IO invests in SDKs. The following is derived from her key points:

Better API design through SDKs

A best practice for developer teams is to implement at least one SDK for the APIs they build. This way, during the SDK development, they will experience themselves how easy or hard it is to implement their API. This can help expose bugs or hidden complexity.

Full code coverage

Customers will only implement the API functions they need for their application, an SDK implemented by your API team by contrast can create an API client with full coverage. The resulting SDK will be more useful for your community and will help expose bugs that might otherwise not be found.



Example of an SDKs page (Apigee)

SDKs improve the developer experience

Developers just want to get the functionality they seek with as little hassle as possible. It is obviously much easier for them to work with a language or platform that they are already familiar with, that way they don't even need to understand how your API works. Besides this obvious benefit, SDKs also help developers circumvent typical Developer Experience (DX) problems.

One important example is authentication, which is one of the biggest stumbling blocks when implementing an API. OAuth issues are often cited as a crucial DX problem. While this problem

[88] http://www.slideshare.net/taylorsoitgoes/creating-a-great-developer-experience-through-sdks"

can be alleviated with good documentation, an SDK can help you completely sidestep this problem, allowing developers to use the built-in authentication. An SDK can also implement error handling for your API, which can be a massive boon during debugging.

SDKs demonstrate best practices

For complex APIs, SDKs can help demonstrate best practices to developers. Even if they don't end up using your SDK, developers can see how your APIs are tied together, and how you expect developers to use them.

What types and how many SDKs do you need?

Community SDKs

GitHub enables developers to build and publish an open source community SDK for your APIs.

On first sight that might seem like a great deal: it can be a lot of work to create a good SDK. Not having to pay for the initial development and maintenance saves a lot of costs and if your API becomes very popular this might be a viable strategy. Some companies that have open sourced their application, consciously don't invest in SDKs, and instead expect the community to give back to their platform.

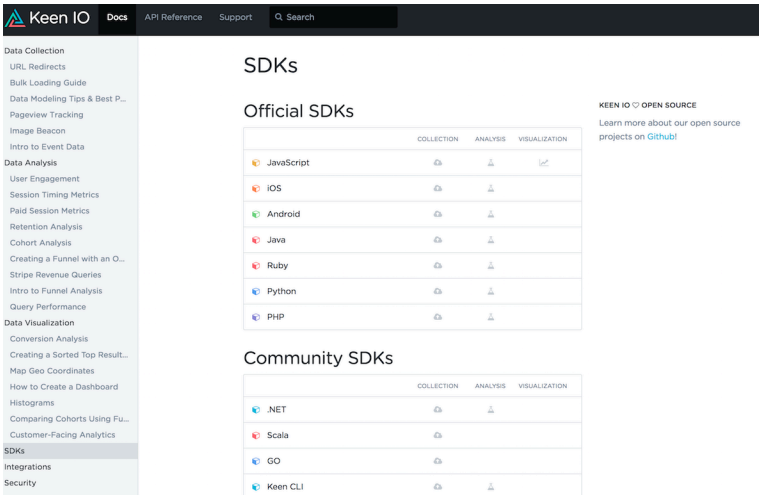
There is, however, a downside. In her talk on SDKs[89], Taylor Barnett advises that it is better to make what she calls "product" SDKs. She also explains why it is important to clearly differentiate between Product and Community SDKs:

1. To indicate the trustworthiness and the expected longevity of an SDK.
2. To explain differences in the developer experience: community SDKs might not follow all best practices and will probably not be as well documented.
3. To set proper maintenance and support expectations. Even if you make the distinction, some community SDK issues will inevitably be submitted through your portal's support channels, not addressing them will damage your brand, but if you don't own the code and if you don't have commit rights this might be difficult.

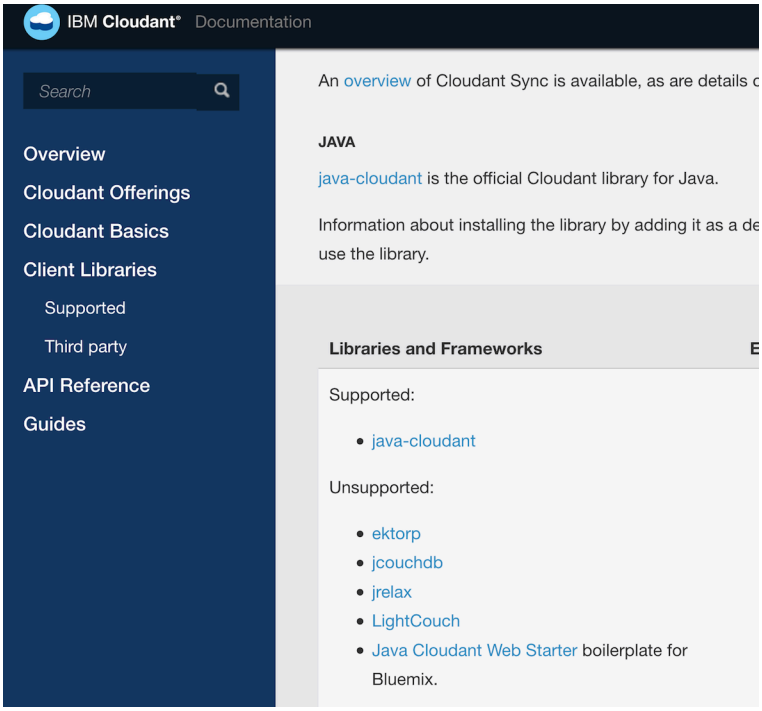
So while it might be tempting to rely on your community to create and maintain open source SDKs, doing so is a form of technical debt. Community SDK maintainers often disappear, or become upset about not being paid while you benefit from their work. As a result your customers might end up integrating with an older version of your API, unaware of best practices, and get frustrated when an SDK doesn't function properly.

In our research sample, Keen IO, Twilio, Asana, DigitalOcean, CenturyLink, IBM Cloudant, Apigee and Dwolla categorized their SDKs according to maintenance status and/or ownership (product/official/supported vs community/third-party).

[89] http://www.slideshare.net/taylorsoitgoes/creating-a-great-developer-experience-through-sdks"



Example of SDKs categorized in a product ("official") and a community section. The scope of each SDK is also indicated (Collection, Analysis, Visualization) (Keen IO)



Example of client library types: supported, unsupported, and third party libraries (IBM Cloudant)

Automatic SDK generation

There is a 3rd option besides community and product SDKs: it is also possible to automatically generate SDKs. E.g. APIMATIC[90] is a service that automatically generates SDKs, tests, code samples and documentation. If your API is not too complex, and you don't have the people or resources to make handcrafted product SDKs, this might be worth exploring.

[90] https://apimatic.io/

There is, however, a caveat: while automatically generated SDKs save a lot of time and money, and even remove some of the release timing issues caused by sequential development, they lose a lot of the DX benefits that product SDKs give. Without creating at least one SDK, your team won't be able to have immediate feedback on their API design. For the time being, machines also lack the required intelligence of a human developer to extrapolate between the best practices of your API and the programming language or platform the SDK is developed for.

How many SDKs should you create?

It makes sense to split SDKs into functional groups, e.g. to make the distinction between data capture, processing, and visualization: a developer might only need part of this functionality e.g. to integrate with their frontend application. This also means that some parts of your APIs might have SDK coverage in one language and not in another. It is not always straightforward what languages/platforms to build SDKs for, and it might take some investigation to figure out what would be good developer communities to target. **In any case it is a good idea to track the usage of your API, if possible in conjunction with business metrics across different SDKs**. This allows you to analyse what communities are providing you a better income to API calls ratio, so you can maximize your growth and profitability.

In our research sample we found that in the range of published SDKs, the number of product SDKs / community supported SDKs greatly varied from portal to portal.

Overview of Libraries and Platform SDKs (as listed on the portals' SDK pages):

Twilio

Product SDKs		*Community SDKs*
Server-Side SDKs:	C# Java Node.js PHP Python Ruby Salesforce	**Smalltalk** (REST API Helper) **Twilio Django helper** **Go** (REST API Helper) **Erlang** (REST API Helper)
Javascript SDKs:	Twilio Client (VoIP) IP Messaging Video TaskRouter	
iOS SDKs:	Twilio Client (VoIP) IP Messaging Video	
Android SDKs:	Twilio Client (VoIP) IP Messaging Video	
Authy SDKs:	Authy Mobile SDK	

Keen IO

Product SDKs (Official SDKs)			*Community SDKs*		
Collection	**Analysis**	**Visualization**	**Collection**	**Analysis**	**Visualization**
Javascript	Javascript	Javascript	.NET	.NET	
iOS	iOS		Keen CLI	Keen CLI	
Android	Android		Scala		
Java	Java		Go		
Ruby	Ruby		Perl		
Python	Python		Arduino		
PHP	PHP		Squirrel		
			CC3200		

IBM Cloudant

Product SDKs	***Community SDKs***
Supported client libraries: **Mobile**: • Cloudant Sync - Android / JavaSE • Cloudant Sync - iOS (CDTDatastore) **Java**: java-cloudant **Node.js**: • nodejs-cloudant • sag-js • nano. • restler. • cradle • cane_passport - Cloudant Angular Node Express with Bootstrap. • Express-cloudant • Node.js Cloudant DB Web Starter • Mobile Cloud **Python**: python-cloudant **Swift**: swift-cloudant **Apache Spark**: spark-cloudant **Objective-C**: objective-cloudant	**Unsupported client libraries:** **Java**: - ektorp - jcouchdb - jrelax - LightCouch - Java Cloudant Web Starter **Third-party libraries:** **C# / .NET**: • MyCouch • LoveSeat • Divan • Relax • Hammock • EasyCouchDB • WDK.API.CouchDB from Kanapes IDE. • CRUD **PHP**: • sag • Doctrine CouchDB Client • PHP-on-Couch **Javascript**: • Backbone.cloudant • sag.js • PouchDB **Ruby**: CouchRest **Meteor**: cloudant:couchdb

Digital Ocean

Product SDKs	*Community SDKs*
Go: Official Go Wrapper **Ruby**: Official Ruby Wrapper	**Ruby**: Barge **Python**: • python-digitalocean • dopy • digitalocean-api **Node**: • dropletapi • do-wrapper **Java**: digitalocean-api-java **PHP**: • Laravel-DigitalOcean • DigitalOceanV2 **Scala**: digitalocean **Clojure**: digital-ocean **Haskell**: digital-ocean **.NET**: DigitalOceanAPI **Objective-C**: DigitalOcean iOS SDK **Perl**: WebService::DO

Asana

Product SDKs	*Community SDKs*
JavaScript (Node): node-asana **JavaScript (Browser)**: node-asana **Python**: python-asana **Ruby**: ruby-asana **Java**: java-asana	Asana refers to GitHub for other platforms

CenturyLink

Product SDKs	***Community SDKs***
Java: CLC Java SDK **.NET**: CLC .Net SDK **Python**: CLC Python SDK **NodeJS**: CLC Node.js SDK **Go**: • CLC Go SDK • Orchestrate Go SDK	

Dwolla

Product SDKs	***Autogenerated SDKs***
Node: dwolla-v2-node **Ruby**: dwolla-swagger-ruby **Python**: dwolla-swagger-python	**PHP**: dwolla-swagger-php **Java**: dwolla-swagger-java

Apigee

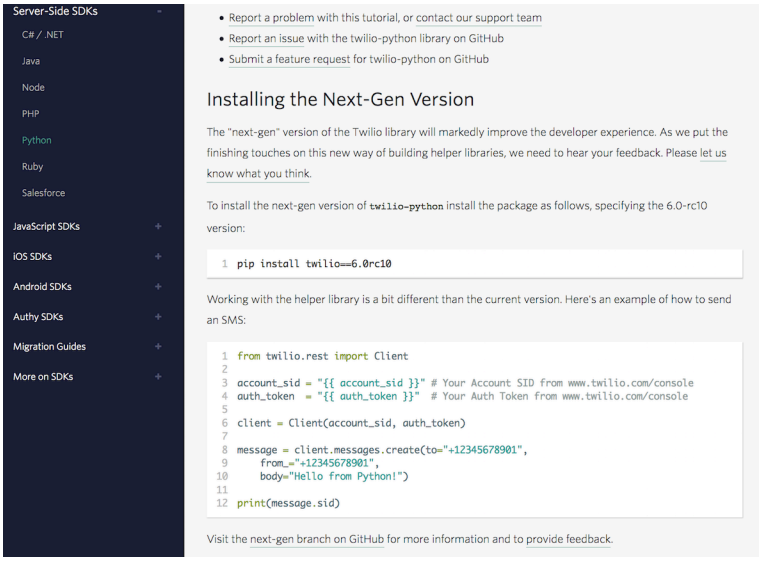
Product SDKs	***Community SDKs***
API BaaS SDKs: **iOS** **Android** **JavaScript** **Node.js Module** **Ruby** **.NET**	

Where are SDKs included in a developer portal's information architecture?

The overview page (frontpage) of 6 developer portals provided links to SDKs in their header, footer or body section. Mashape referred to their Unirest libraries in the sidebar menu of its overview

page. 2 portals (Asana and Apigee) included their SDKs in the hierarchical sidebar menu on their documentation pages.

Mashape set up a separate page for its Unirest product, a general purpose library that developers can use to simplify HTTP REST requests. So you could argue that this should not be categorized as a product SDK. The other portals mainly listed their code on GitHub[91], which is developer-friendly, free for open source projects, and has (at least for the time being) become a de facto standard in the developer community. One company used both GitHub and Google code.



Example of code samples on-site and a reference to GitHub for more information (Twilio)

The developer portals in our sample all provided a list of available SDKs.

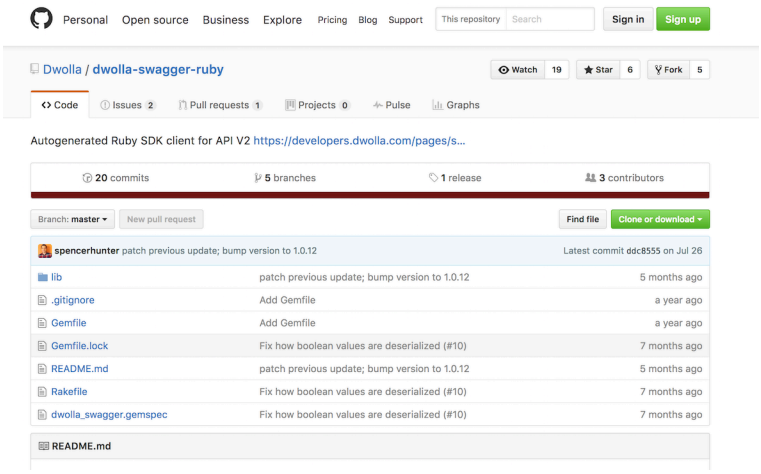
GitHub is an open platform that doesn't have any barriers that prevent developers from adding community SDKs. It is exactly this permissiveness of GitHub and similar platforms that enables the community SDK phenomenon: allowing API owners and their customers to build further upon the work of other developers that they otherwise might never have access to.

It is however crucial to establish a minimum of community management processes to support and recognize the work of outside developers. API owners need to have a discovery and curation process that helps them identify new community SDKs that need to be evaluated and described so they can be added to the SDK listing on a developer portal.

When community SDKs are not listed on a developer portal, it can be time-consuming for developers to find out what product / community SDKs are available for an API.

Developer portals need to make it as easy as possible to discover, evaluate, and select an appropriate SDK. Depending on the number of APIs you provide, how related they are, and their complexity you will need to provide tools to help developers navigate your SDK offering.

[91] https://github.com/



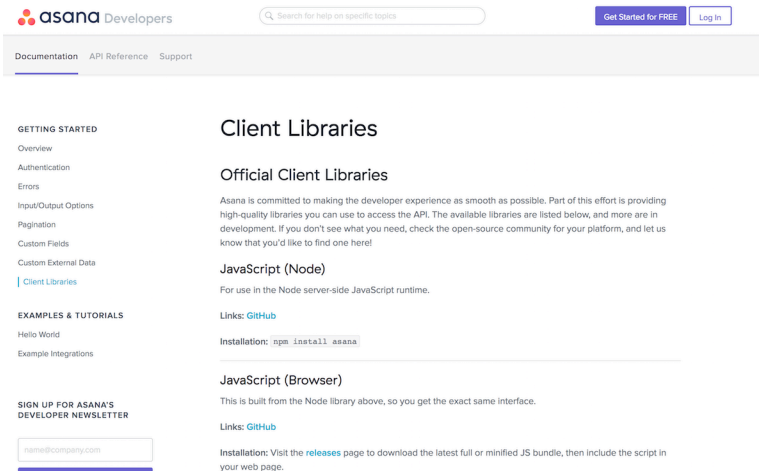
Example of code on GitHub (Dwolla)

Onboarding with SDK documentation

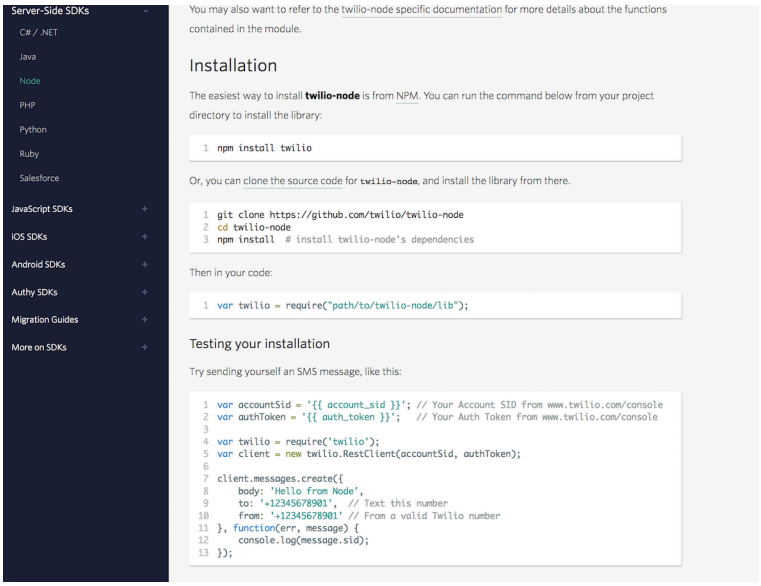
If you want to make it as easy as possible for a developer to get started with your API, you could provide platform specific onboarding documentation. An SDK then becomes a part of your onboarding journey. This allows developers that want to use your API to select their platform, and get instructions on how to develop with your SDK instead of your API. They don't need to switch into an API context and can stay in the context of their platform instead.

If your SDK has any dependencies on third party code, those can become a major DX issue for developers. Taylor Barnett calls it the "Scary world of dependencies" and recommends that SDK developers:

- Carefully evaluate what dependencies to choose when there are multiple options, to make it as easy as possible for your target communities.
- Address dependencies in the onboarding documentation.
- Pay special attention to any changes in dependencies between SDK versions.



Example of libraries in the Getting Started section of a developer portal (Asana)



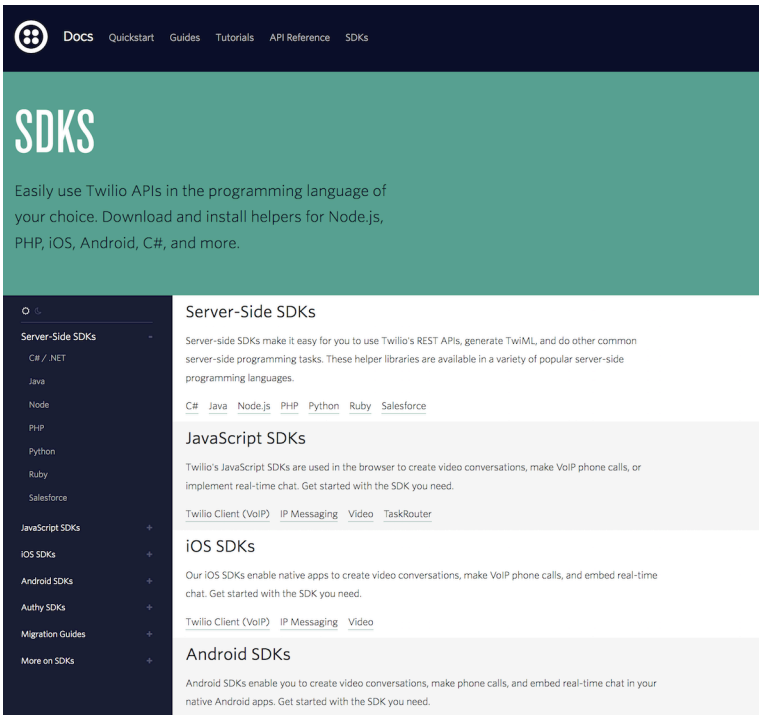
Example of an onboarding guide with SDKs (Twilio)

How are SDKs exposed?

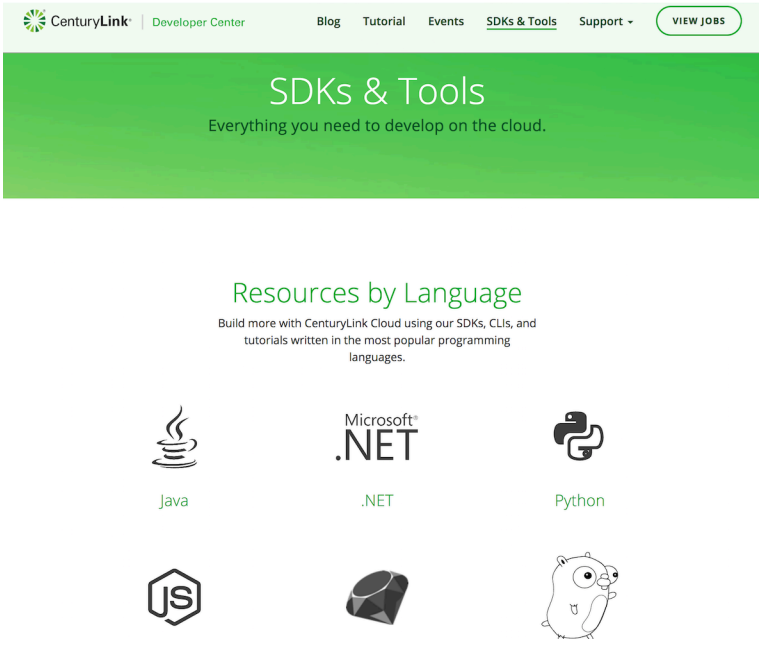
Developer portals implemented:

- Categorization into one, two or three columns that follow a logical grouping for the SDKs so that developers can understand faster where they can find the SDK they need.
- Icons to make it easier to recognise code languages and platforms.
- CTAs (Call to Actions like “View Libraries”, “See the source on GitHub”) that link to the code repository.

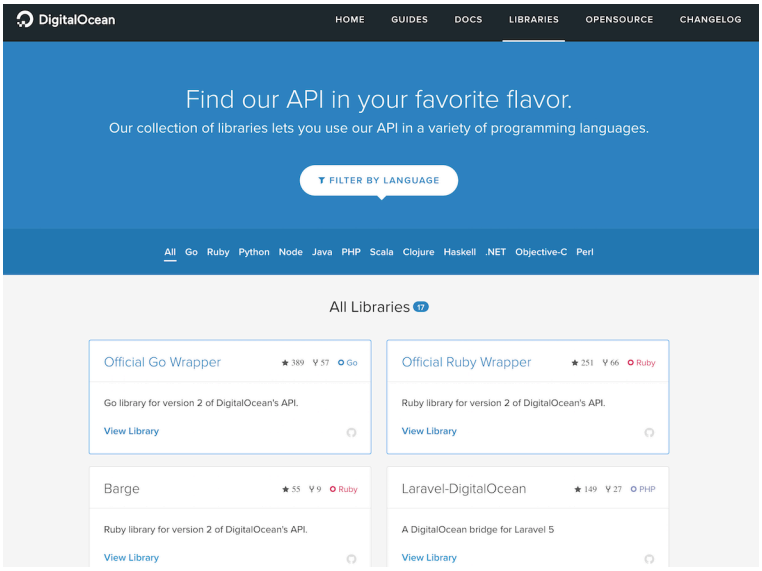
- Headlines (“Find our API in your favorite flavor” - DigitalOcean) to engage users.
- Filtering options (CTAs like “Filter by language” or “Resources by language” or via a hierarchical sidebar menu) for easy content filtering.
- Labelling, like “Product SDKs” and “Community SDKs” (Keen IO) to set proper expectations and to make it clear what the source of an SDK is.
- Visual design elements e.g. change the text and library border color, to make a distinction between product and community SDKs (DigitalOcean).



Example of SDKs with a hierarchical sidebar menu (Twilio)



Example of an SDK page where the languages are accompanied by their icons (CenturyLink)



Example of all API Libraries, with language selector, categorized into two columns. The text and table border distinguishes between "official" product and community libraries (DigitalOcean)

Labels

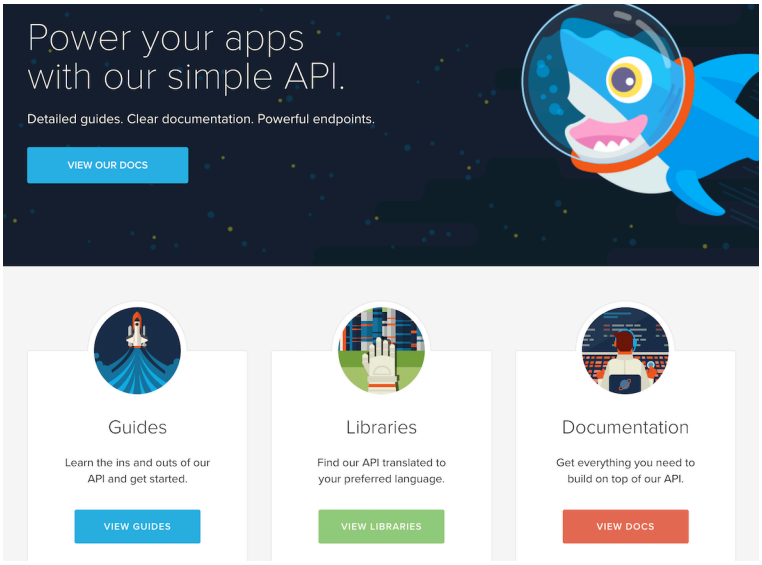
Developer portals applied the following labels to refer to SDKs:

- SDKs (mentioned on 5 portals)

- Libraries (4)
- Client Libraries (2)
- Helper Libraries (2)
- Helpers (1)
- Libraries and Frameworks (1)

Some developer portals applied various labels to identify SDKs:

- SDKs / Libraries / Helper Libraries / Helpers (Twilio)
- Client Libraries / Libraries / Libraries and Frameworks (IBM Cloudant)
- SDKs / Helper Libraries (Dwolla)



Example of SDKs labelled as "Libraries" (DigitalOcean)

Summary: SDKs on developer portals - best practices

Providing practical examples through code improves DX: it can help developers to learn from existing examples, to onboard easily and save implementation time. The following are key tips on how to include SDKs into a portal's documentation:

- Be consistent in terminology (choose one word to describe the SDKs)
- Choose a code repository that enables community contributions - in our sample, GitHub was by far the most popular choice
- Include multiple code languages and platforms to target different developer communities

- Add filtering options to make it easier for developers to find the proper SDK
- Make the code overview pages visually attractive: add icons, columns, headlines
- Separate product and community SDKs (Taylor Barnett)
- Measure usage for your SDKs, so you can gain insights about your customer communities (Taylor Barnett)
- Write the SDK documentation first and include sections for troubleshooting and changelog/release notes - this will help you evaluate the developer experience (Taylor Barnett)
- Keep it native: start with the languages that are the most popular for your audience and that the documentation team is familiar with (Taylor Barnett)

Many thanks to Taylor Barnett from Keen IO for her really insightful talk about SDKs[92], we leaned heavily on her presentation for this chapter!

[92] http://www.slideshare.net/taylorsoitgoes/creating-a-great-developer-experience-through-sdks

Want more?

You have reached the end of *Developer Portals Components. API Documentation Patterns Part I*. We hope you enjoyed it! We are already planning two more white papers about documentation patterns, one about strategies and one on technologies. To get them first, and to keep informed about our research, we absolutely recommend you to **subscribe to our Developer portal mailing list**[93]!

If you have any questions, would like to ask for further information or share your thoughts on developer portals, or if you have any feedback on the white paper, you can reach the authors at developerportals@pronovix.com.

Thank you for reading our work!

The authors.

Kristof Van Tomme

Kathleen De Roo

[93] http://eepurl.com/bF0ztz

About the authors

Kristof Van Tomme

Co-founder, CEO

Kristof Van Tomme is an open source strategist and architect. He is the CEO and co-founder of Pronovix, a company that builds developer portals and documentation systems in Drupal. He's got a degree in bioengineering and is a regular speaker at technology conferences. For a few years now he's been building bridges between the documentation and Drupal community. He shares his time between Belgium where he lives, Hungary where Pronovix has its office, and London where he started the local Write the Docs meetup.

Kathleen De Roo

Copywriter

As a copywriter and member of the content team, Kathleen is responsible for writing, reviewing and editing website copy and blog posts. She's got an interest in information architecture. She holds master's degrees in History and in Archiving / Records Management. Before joining Pronovix, she gained professional experience in teaching and office management and was a volunteer for several non-profit organizations.

Contributors

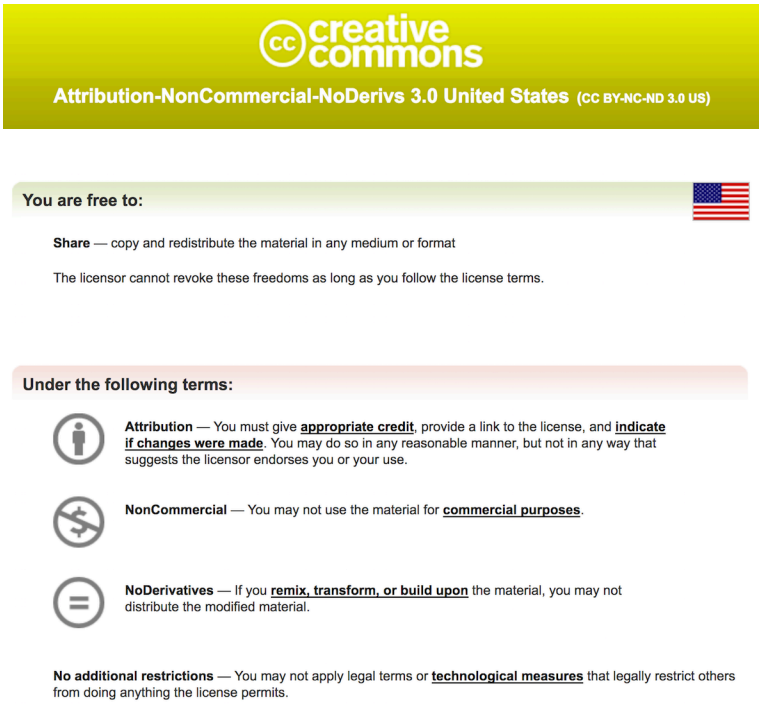
The authors would like to thank:

- **Laura Vass** and **István "Steve" Szabó**, fellow members of the company's content team, for their advice, help behind the scenes and constant support.
- UX manager **Kata Nagygyörgy**, for helping ideate the research.
- Developers **Tamás Demeter-Haludka** and **László Csécsy**, for explaining technical terminology.
- **Michael Meng and his team**, **Taylor Barnett** and **Jan Christian Krause** for their inspiring talks.

Creative Commons License

Would you like to share our work? Thank you!

May we ask you to keep **this Creative Commons License**[94] in mind while you do?



Creative Commons License

[94] https://creativecommons.org/licenses/by-nc-nd/3.0/us/