Welcome to

a//things.api

meet the speakers

Ramana Lashkar / Principal Engineer
Tyler Toth / Software Engineer
Paris Roman / UI/UX Engineer

a//things.api

purpose & agenda

purpose

All Thing API meetup is for API practitioners, implementers and developers to come together to share their experience and learnings on defining, designing, building, launching, managing and consuming APIs based on REST/SOAP/GraphQL/JSON/XML technologies.

agenda

- What is Authentication and Different types of Authentication
- Oauth1 and Oauth2 Authentication flow
- OpenAPI specification and API Documentation
- Swagger
- Live Examples of Authentication and OpenAPI/Swagger-UI

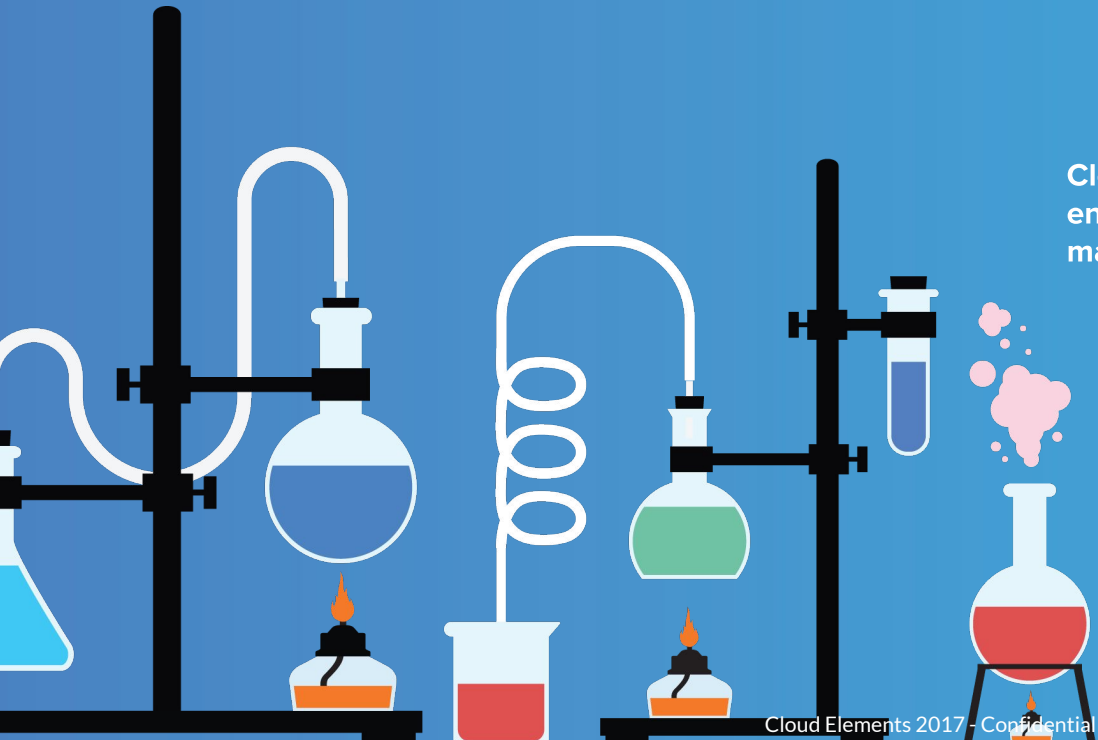
a//things.api

cloud elements

Who we are

Cloud Elements

Cloud Elements is a cloud API integration platform that enables developers to publish, integrate, aggregate and manage all of their APIs through a unified platform.



cloud elements

MAKING THE APIS YOU USE WORK TOGETHER

Embed integration right into your application.

Offer integrations that work right **out-of-the-box** for your customers.

cloud elements


WRITING TO THE API
Authenticate
Discover Custom Objects
Map & Transform
Events Workflows
Synchronize
Log
Bulk

OUR HUB MODEL

ONE-TO-MANY INTEGRATIONS

UNIFIED INTEGRATION WITH OUR

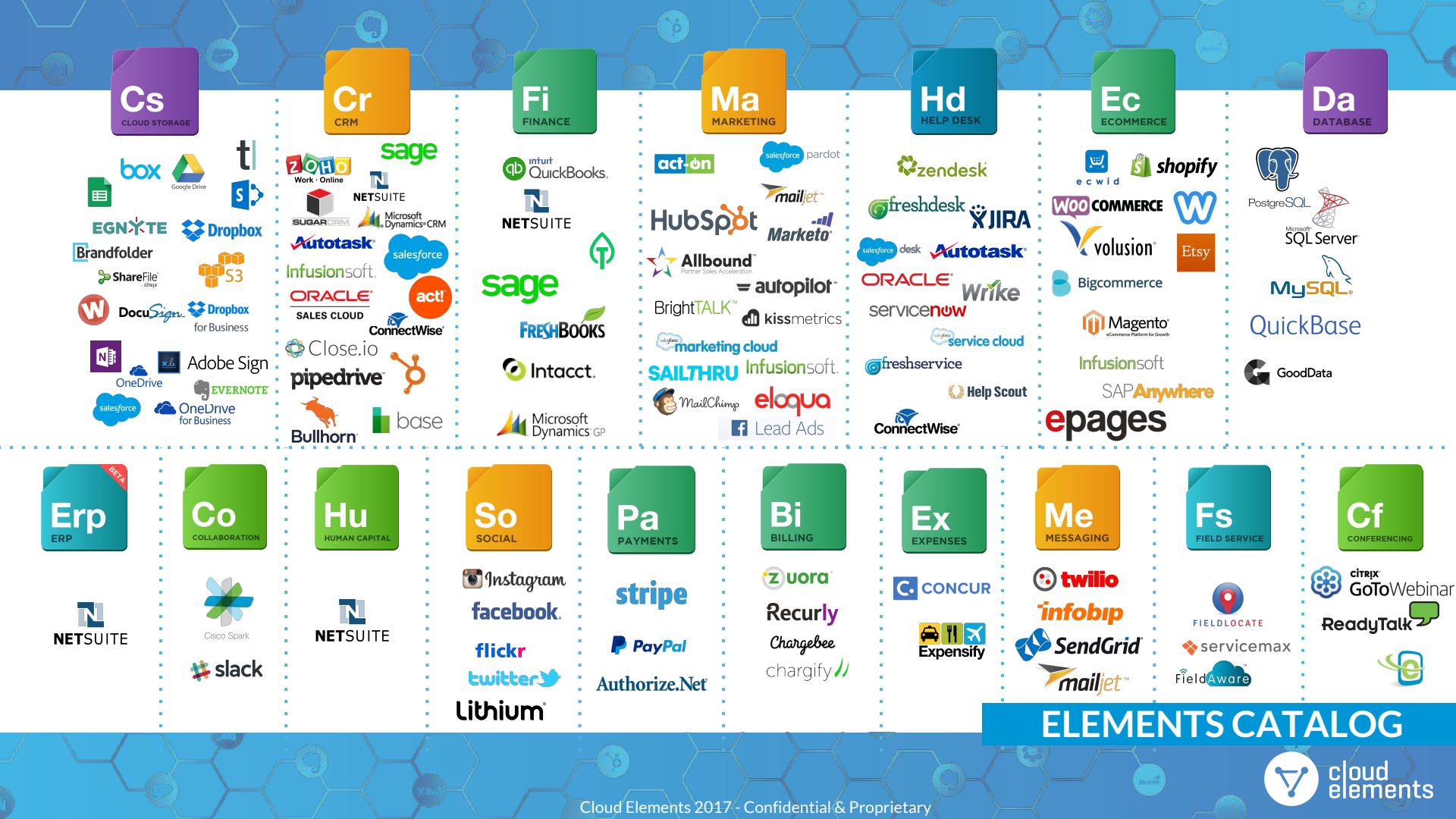
HUB APIs

You integrate with our Hub APIs to connect with all services in that category

This reduces development cost, by delivering many integrations through a unified API

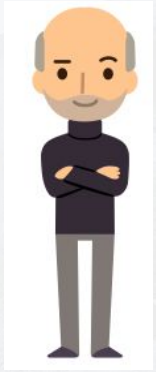
We handle the maintenance and roadmap for each element with minimal cost to you

cloud elements

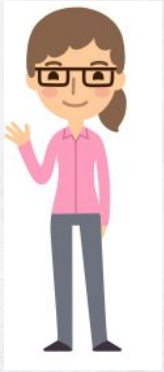

Cs
CLOUD STORAGE
box
Google Drive
EGNYTE
Dropbox
Brandfolder
ShareFile
S3
DocuSign
Dropbox for Business
OneDrive
Adobe Sign
EVERNOTE
salesforce
OneDrive for Business
Cr
CRM
ZOHO Work · Online
sage
NETSUITE
SUGARCRM
Microsoft Dynamics CRM
Autotask
salesforce
Infusionsoft
ORACLE SALES CLOUD
act!
ConnectWise
Close.io
pipedrive
Bullhorn
base
Fi
FINANCE
intuit QuickBooks
NETSUITE
sage
FRESHBOOKS
Intacct.
Microsoft Dynamics GP
Ma
MARKETING
act-on
salesforce pardot
mailjet
HubSpot
Marketo
Allbound
autopilot
BrightTALK
kissmetrics
marketing cloud
SAILTHRU
Infusionsoft
MailChimp
eloqua
Lead Ads
Hd
HELP DESK
zendesk
freshdesk
JIRA
salesforce desk
Autotask
ORACLE
Wrike
servicenow
service cloud
freshservice
Help Scout
ConnectWise
Ec
ECOMMERCE
ecwid
shopify
WOO COMMERCE
volusion
Etsy
Bigcommerce
Magento eCommerce Platform for Growth
Infusionsoft
SAP Anywhere
epages
Da
DATABASE
PostgreSQL
Microsoft SQL Server
MySQL
QuickBase
GoodData
Erp
ERP
BETA
NETSUITE
Co
COLLABORATION
Cisco Spark
slack
Hu
HUMAN CAPITAL
NETSUITE
So
SOCIAL
Instagram
facebook
flickr
twitter
Lithium
Pa
PAYMENTS
stripe
PayPal
Authorize.Net
Bi
BILLING
zuora
Recurly
Chargebee
chargify
Ex
EXPENSES
CONCUR
Expensify
Me
MESSAGING
twilio
infobip
SendGrid
mailjet
Fs
FIELD SERVICE
FIELDLOCATE
servicemax
FieldAware
Cf
CONFERENCING
CITRIX GoToWebinar
ReadyTalk
ELEMENTS CATALOG
cloud elements

identify yourself

sample personas



Dave, The Developer
Your raw, unedited API is targeted directly at Dave the Developer. He discovers new APIs through developer communities and builds his own integrations.



Izzy, IT Manager
Izzy knows the in's and out's of her company's apps, as each system that is purchased, Izzy works with to keep connected, sharing data seamlessly. Izzy is the technical integrator.

Pete, Product Manager
As a product manager, Pete is responsible for connecting business needs with market demands, managing priorities, and meeting with customers.

authentication

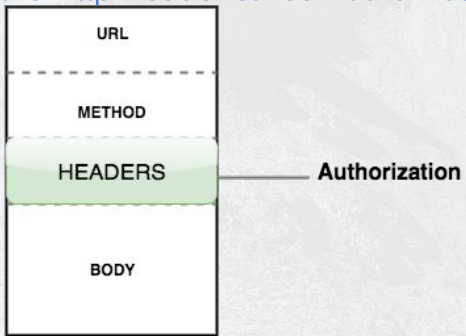
What is Authentication ? Why do you need one ?

With API we know how to talk to the server, though, leaves an important question: how does the server know the client is who it claims to be?

Logging-in with a username and password is one example of a technical process known as authentication.
When you authenticate with a server, you prove your identity to the server by telling it information that only you know. Once the server knows who you are, it can trust you and send you the private data in your account.

There are several techniques APIs use to authenticate a client. These are called authentication schemes.

Most Authentication schemes use the Http Header called Authorization



authentication schemes

- Basic Authentication
- API Key Authentication
- Oauth1
- Oauth2
- Custom Authentication

Basic Auth

The logging-in using username/password is the most basic form of authentication. In fact, the official name for it is Basic Authentication

Widely used protocol for simple username/ password

Provides no confidentiality protection for the transmitted credentials.

How it works:
The client takes these two credentials, smooshes them together to form a single value and passes that along in the request in an HTTP header called Authorization.

ex: username: meetup and password: allthingsapi
then the Authorization header value will be *Basic bWVldHVwOmFsbHRoaW5nc2FwaQ==*

When the server receives the request, it looks at the Authorization header and compares it to the credentials it has stored. If the username and password match one of the users in the server's list, the server fulfills the client's request as that user. If there is no match, the server returns a special status code (401) to let the client know that authentication failed and the request is denied.

API Key
authentication

Doesn't require shared credentials

Requires API to be accessed by a unique key comprised of numbers and letters

The flexibility is there with API Key authentication to limit control as well as protect user passwords.

API key authentication is a bit like the wild west; everybody has their own way of doing it.

Typically goes in the Authorization header in lieu of username and password.

Another approach is to add the key onto the URL (http://example.com?api_key=my_secret_key).

Less common is to bury the key somewhere in the request body next to the data. Wherever the key goes, the effect is the same - it lets the server authenticate the client.

ex: c2RhZnNkZnNkYWZhc2RmYXNkZnNkYWZhc2RmYXNkZ

Issue: Communicate the long keys to the users

Oauth

Automating the key exchange is one of the main problems OAuth solves.

It provides a standard way for the client to get a key from the server by walking the user through a simple set of steps

From the user's perspective, all OAuth requires is entering credentials. Behind the scenes, the client and server are chattering back and forth to get the client a valid key.

There are currently two versions of OAuth, aptly named **Oauth1** and **Oauth2**. Understanding the steps in each is necessary to be able to interact with APIs that use them for authentication.

oauth2

Who are involved

The User - A person who wants to connect two websites (applications) they use

The Client - The website(application) that will be granted access to the user's data

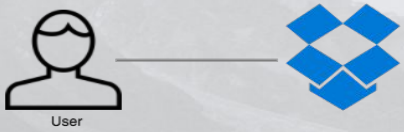
The Server - The website(application) that has the user's data

OAUTH2

Step 1 - User Tells Client to Connect to Server (Dropbox)

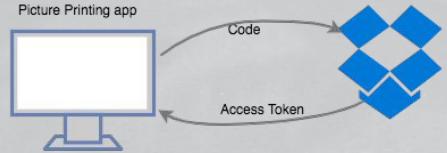
Step 2 - Client Directs User to Server (Dropbox)

Step 3 - User Logs-in to Server(Dropbox) and Grants Client Access



Step 4 - Server(Dropbox) Sends User Back to Client, Along with Code

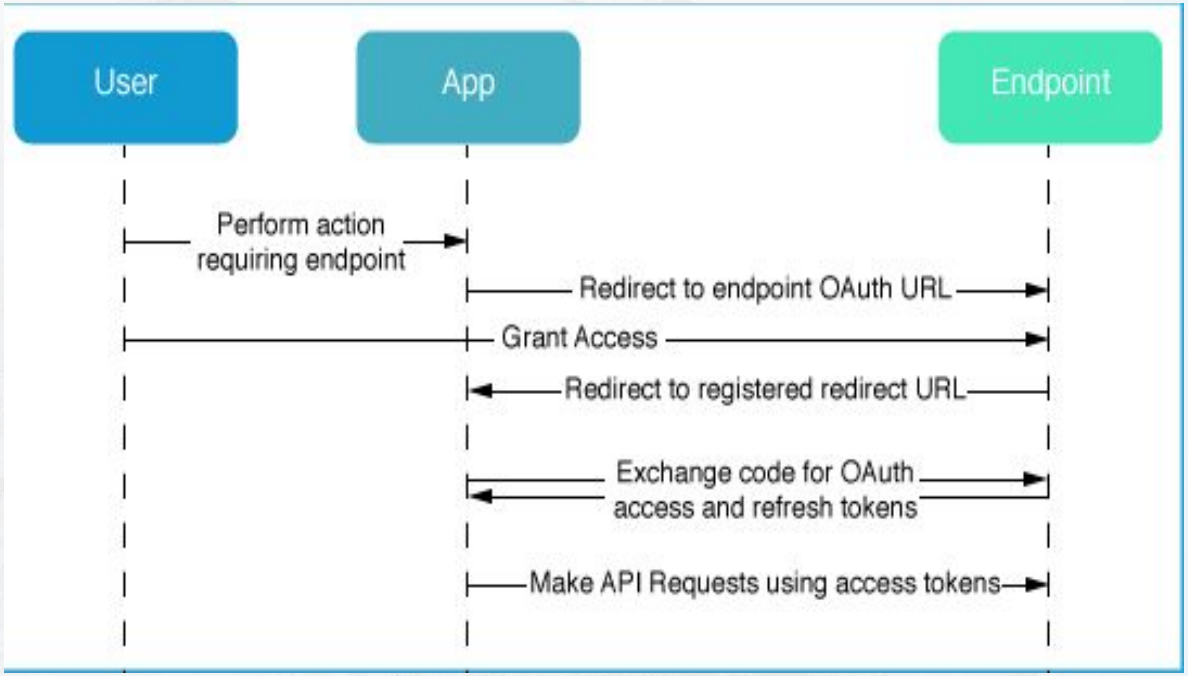
Step 5 - Client Exchanges Code + Secret Key with server(Dropbox) for Access Token



Step 6 - Client Fetches Data from Server(Dropbox)

Client Refreshes Token (Optional)

oauth2



oauth2

Directing the user to a URL like the following:

```
https://www.dropbox.com/1/oauth2/authorize?client_id=<app
key>&response_type=code&redirect_uri=<redirect URI>&state=<CSRF token>
```

After the user has authorized your app, they'll be sent to your redirect URI, with a few query parameters:

```
https://www.photoprintingapp.com/mycallback?code=<authorization code>&state=<CSRF token>
```

Obtain an access token

```
curl https://api.dropbox.com/1/oauth2/token -d code=<authorization code> -d
grant_type=authorization_code -d redirect_uri=<redirect URI> -u <app key>:<app secret>
```

Response

```
{"access_token": "<access token>", "token_type": "Bearer", "uid": "<user ID>"}
```

Call the API

```
curl https://api.dropbox.com/1/account/info -H "Authorization: Bearer <access token>"
```

oauth1

What is different than Oauth2

access tokens do not expire.

OAuth 1 includes an extra step. Between Steps 1 and 2, OAuth 1 requires the client to ask the server for a request token. This token acts like the authorization code in Oauth2 and is what gets exchanged for the access token.

Third difference is that OAuth 1 requires requests to be **digitally signed**. Signatures allow the server to verify the authenticity of the requests.
Today, however, most API traffic happens over a channel that is already secure (HTTPS).
Recognizing this, OAuth 2 eliminates signatures in an effort to make version two easier to use.

oauth1

Step 1 - User Tells Client to Connect to Server (Dropbox)

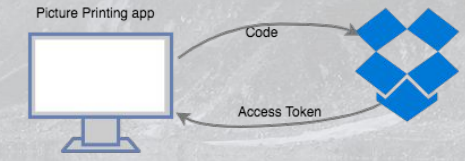
Step 2 - Client Asks server for a Request Token

Step 3 - Client Directs User to Server (Dropbox)

Step 4 - User Logs-in to Server(Dropbox) and Grants Client Access

Step 5 - Server(Dropbox) Sends User Back to Client, Along with Code

Step 6 - Client Exchanges Code + Secret Key with server(Dropbox) for Access Token



Step 7 - Client Fetches Data from Server(Dropbox)

authentication error code

Http Status Code: 401 Unauthorized

Indicating that the request requires authentication. User/agent unknown by the server.

Http Status Code: 403 Forbidden

Indicating the server understood the request but refused to fulfill it. User/agent known by the server but has insufficient credentials.

Lets see it LIVE

Documentation

Documentation ?! Why do I care ?

You've gone through all the work to carefully design your APIs so that they're fast, versatile and easy to use. Next, you need to document them so developers can start using them!

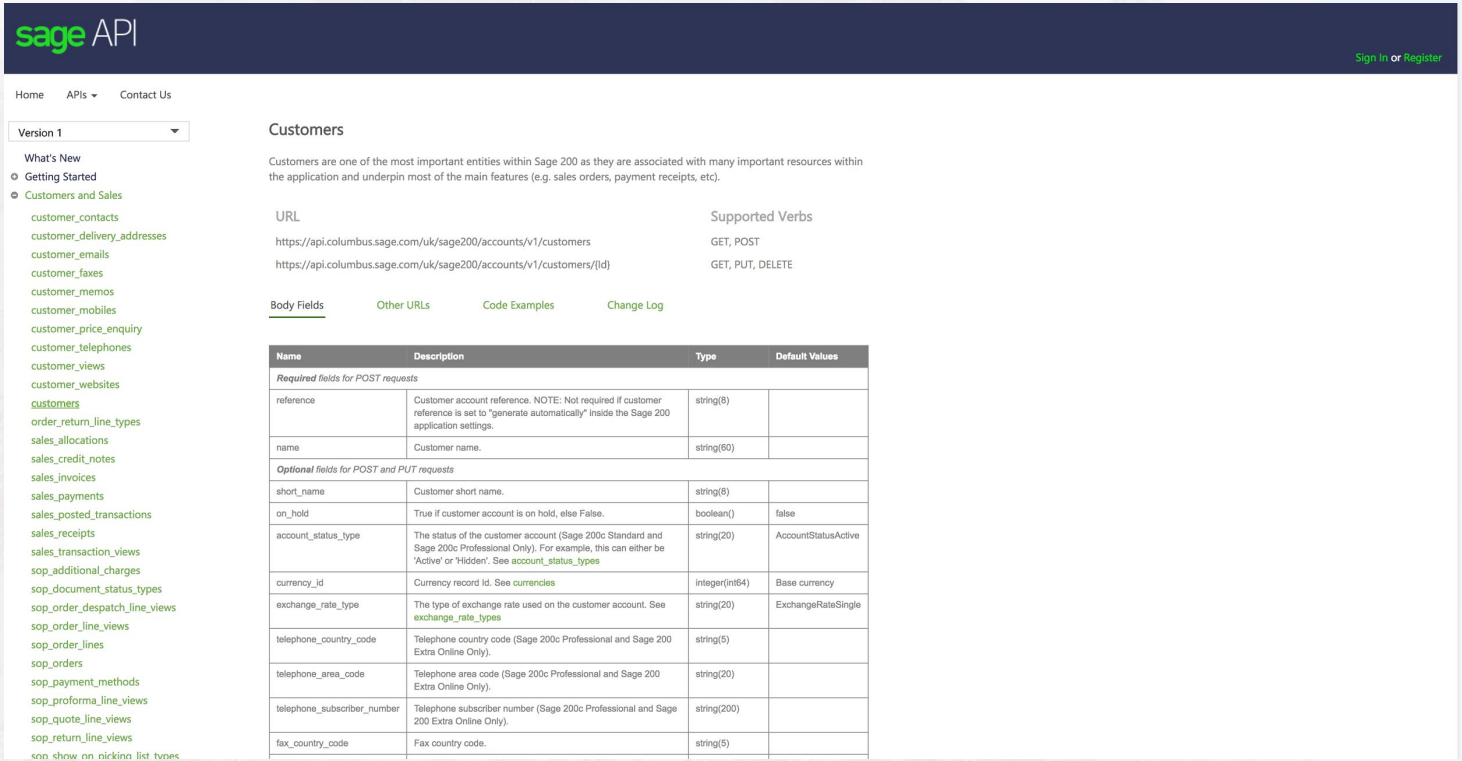
Everyone has their own version of documentation and there have been numerous documentations for RESTful APIs over the years. Namely:

- (...) **Table** - simple table telling you URLs, methods and parameters allowed. Human friendly, but machine illegible.
- (2000) **Web Services Description Language (WSDL)** - XML based interface definition language typically used by SOAP services to describe their APIs. Machine readable, but not human friendly.
- (2009) **Web Services Application Language (WADL)** - XML based interface language, adapted from WSDL, to be used to describe RESTful APIs. Machine readable, but not human friendly.
- (2010) **Open API Specification (Swagger)** - XML/JSON based interface specification used to describe, produce, consume and visualize RESTful Web Services. Machine readable and human friendly - current industry standard
- Countless others (**OData**, **RAML**, **Loopback**, etc.)

documentation

Table

Very human legible, but unable to consume in a automated way



https://developer.columbus.sage.com/docs#/uk/sage200/accounts/v1/customers

documentation

WSDL

Typical documentation for SOAP based webservices

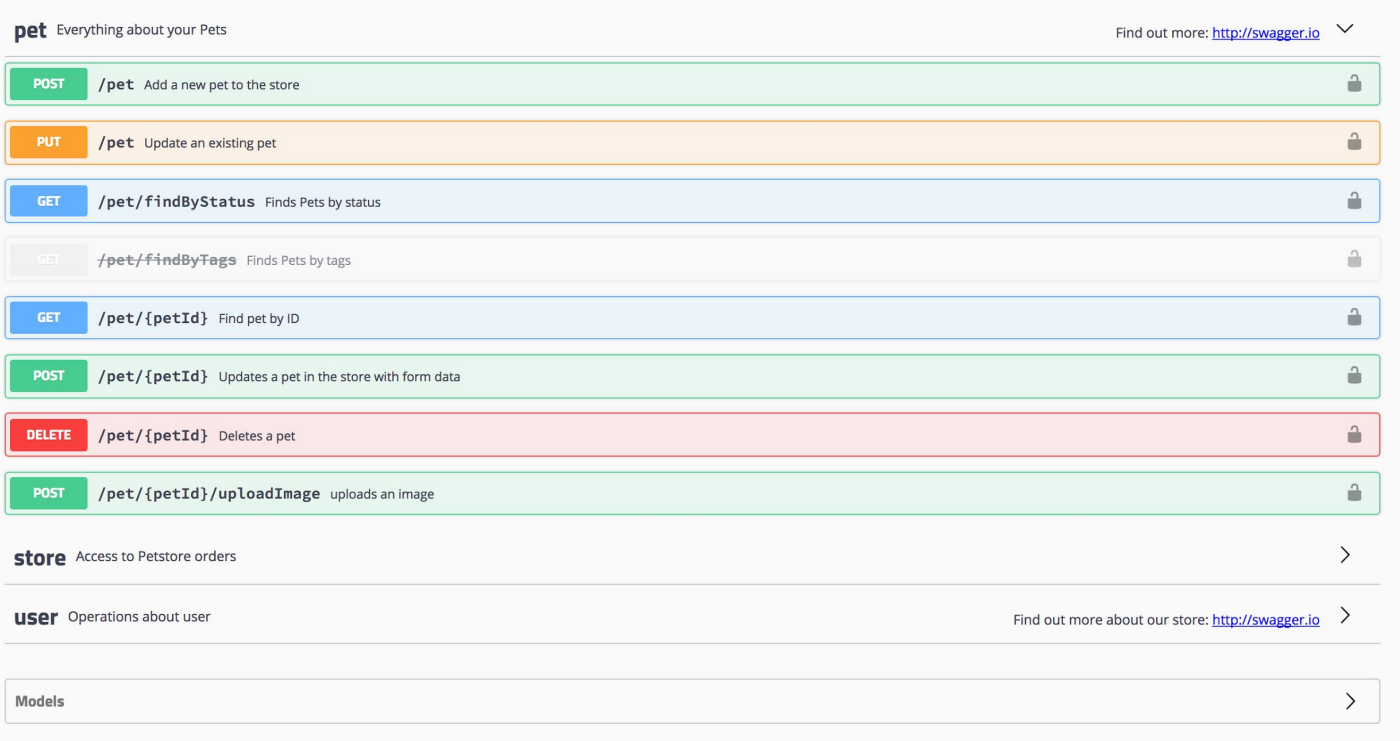
```
<?xml version="1.0" encoding="UTF-8"?>
<description xmlns="http://www.w3.org/ns/wsdl"
             xmlns:tns="http://www.tmsws.com/wsdl20sample"
             xmlns:whttp="http://schemas.xmlsoap.org/wsdl/http/"
             xmlns:wsoap="http://schemas.xmlsoap.org/wsdl/soap/"
             targetNamespace="http://www.tmsws.com/wsdl20sample">

<documentation>
    This is a sample WSDL 2.0 document.
</documentation>

<!-- Abstract type -->
   <types>
      <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
                 xmlns="http://www.tmsws.com/wsdl20sample"
                 targetNamespace="http://www.example.com/wsdl20sample">

         <xs:element name="request"> ... </xs:element>
         <xs:element name="response"> ... </xs:element>
      </xs:schema>
   </types>

<!-- Abstract interfaces -->
   <interface name="Interface1">
      <fault name="Error1" element="tns:response"/>
      <operation name="Get" pattern="http://www.w3.org/ns/wsdl/in-out">
         <input messageLabel="In" element="tns:request"/>
         <output messageLabel="Out" element="tns:response"/>
      </operation>
   </interface>

<!-- Concrete Binding Over HTTP -->
   <binding name="HttpBinding" interface="tns:Interface1"
            type="http://www.w3.org/ns/wsdl/http">
      <operation ref="tns:Get" whttp:method="GET"/>
   </binding>

<!-- Concrete Binding with SOAP-->
   <binding name="SoapBinding" interface="tns:Interface1"
            type="http://www.w3.org/ns/wsdl/soap"
            wsoap:protocol="http://www.w3.org/2003/05/soap/bindings/HTTP/"
            wsoap:mepDefault="http://www.w3.org/2003/05/soap/mep/request-response">
      <operation ref="tns:Get" />
   </binding>

<!-- Web Service offering endpoints for both bindings-->
   <service name="Service1" interface="tns:Interface1">
      <endpoint name="HttpEndpoint"
                binding="tns:HttpBinding"
                address="http://www.example.com/rest/"/>
      <endpoint name="SoapEndpoint"
                binding="tns:SoapBinding"
                address="http://www.example.com/soap/"/>
   </service>
</description>
```

a//things.api

documentation

Open API Specification (Swagger)

Current industry standard for documenting RESTful APIs

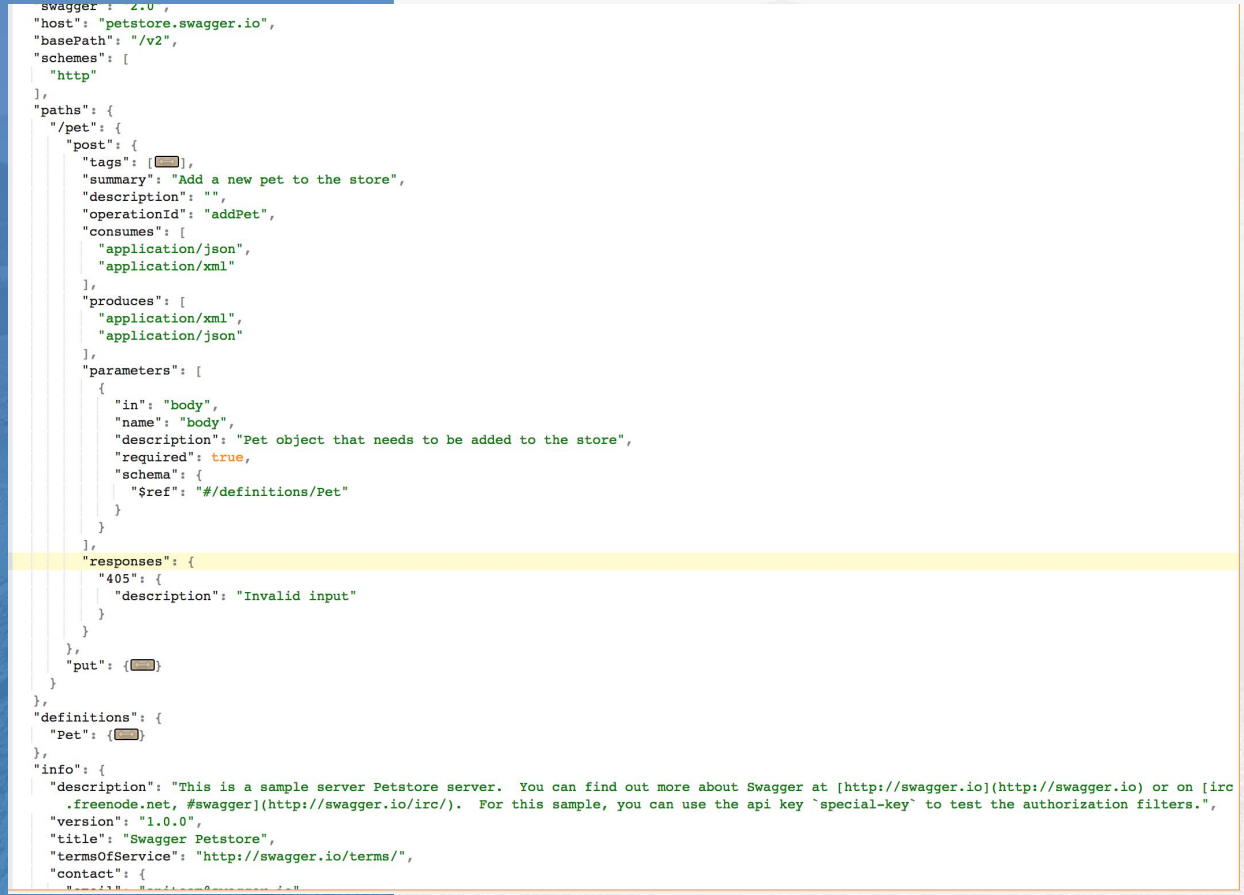


http://petstore.swagger.io/

a//things.api

documentation

Open API Specification (Swagger)



```
"swagger": "2.0",
"host": "petstore.swagger.io",
"basePath": "/v2",
"schemes": [
  "http"
],
"paths": {
  "/pet": {
    "post": {
      "tags": [...],
      "summary": "Add a new pet to the store",
      "description": "",
      "operationId": "addPet",
      "consumes": [
        "application/json",
        "application/xml"
      ],
      "produces": [
        "application/xml",
        "application/json"
      ],
      "parameters": [
        {
          "in": "body",
          "name": "body",
          "description": "Pet object that needs to be added to the store",
          "required": true,
          "schema": {
            "$ref": "#/definitions/Pet"
          }
        }
      ],
      "responses": {
        "405": {
          "description": "Invalid input"
        }
      }
    },
    "put": {...}
  }
},
"definitions": {
  "Pet": {...}
},
"info": {
  "description": "This is a sample server Petstore server.  You can find out more about Swagger at [http://swagger.io](http://swagger.io) or on [irc
    .freenode.net, #swagger](http://swagger.io/irc/).  For this sample, you can use the api key `special-key` to test the authorization filters.",
  "version": "1.0.0",
  "title": "Swagger Petstore",
  "termsOfService": "http://swagger.io/terms/",
  "contact": {
```

Quick reference:

- **paths** : Relative path to the endpoint. Contains method operation, parameter information, etc.
- **info :** Provides metadata about the API
- **definitions :** Contains the field level metadata information about a particular schema

a//things.api

Open API Specification (Swagger)

```
"definitions": {
  "Pet": {
    "type": "object",
    "required": [
      "name",
      "photoUrls"
    ],
    "properties": {
      "id": {
        "type": "integer",
        "format": "int64"
      },
      "category": {
        "$ref": "#/definitions/Category"
      },
      "name": {
        "type": "string",
        "example": "doggie"
      },
      "photoUrls": {
        "type": "array",
        "xml": {
          "name": "photoUrl",
          "wrapped": true
        },
        "items": {
          "type": "string"
        }
      },
      "status": {
        "type": "string",
        "description": "pet status in the store",
        "enum": [
          "available",
          "pending",
          "sold"
        ]
      }
    },
    "xml": {
      "name": "Pet"
    }
  }
}
```

Quick reference:

- **Pet** : Object schema name
- **required :** Array listing the required fields necessary
- **properties :** Lists all the fields available in the **Pet** object
  - **$ref** : denotes the reference to the schema definition
  - **type** : denotes the type of the field

Common Name	`type`	`format`	Comments
integer	`integer`	`int32`	signed 32 bits
long	`integer`	`int64`	signed 64 bits
float	`number`	`float`	
double	`number`	`double`	
string	`string`		
byte	`string`	`byte`	base64 encoded characters
binary	`string`	`binary`	any sequence of octets
boolean	`boolean`		
date	`string`	`date`	As defined by `full-date` - RFC3339
dateTime	`string`	`date-time`	As defined by `date-time` - RFC3339
password	`string`	`password`	A hint to UIs to obscure input.

Lets see it LIVE

Helpful Links

https://blog.cloud-elements.com

https://blog.cloud-elements.com/topic/developer

http://www.vinaysahni.com/best-practices-for-a-pragmatic-restful-api#restful

https://blog.cloud-elements.com/authentication-how-to-use-cloud-elements-authentication-apis

https://resources.cloud-elements.com/api-evangelist/an-openapi-spec-for-a-building-permits-api

https://www.openapis.org/blog

https://oauth.net/2/

https://tools.ietf.org/html/rfc5849

Q&A.