LONESTAR ELIXIRCONF
2018

User Centered

API Versioning

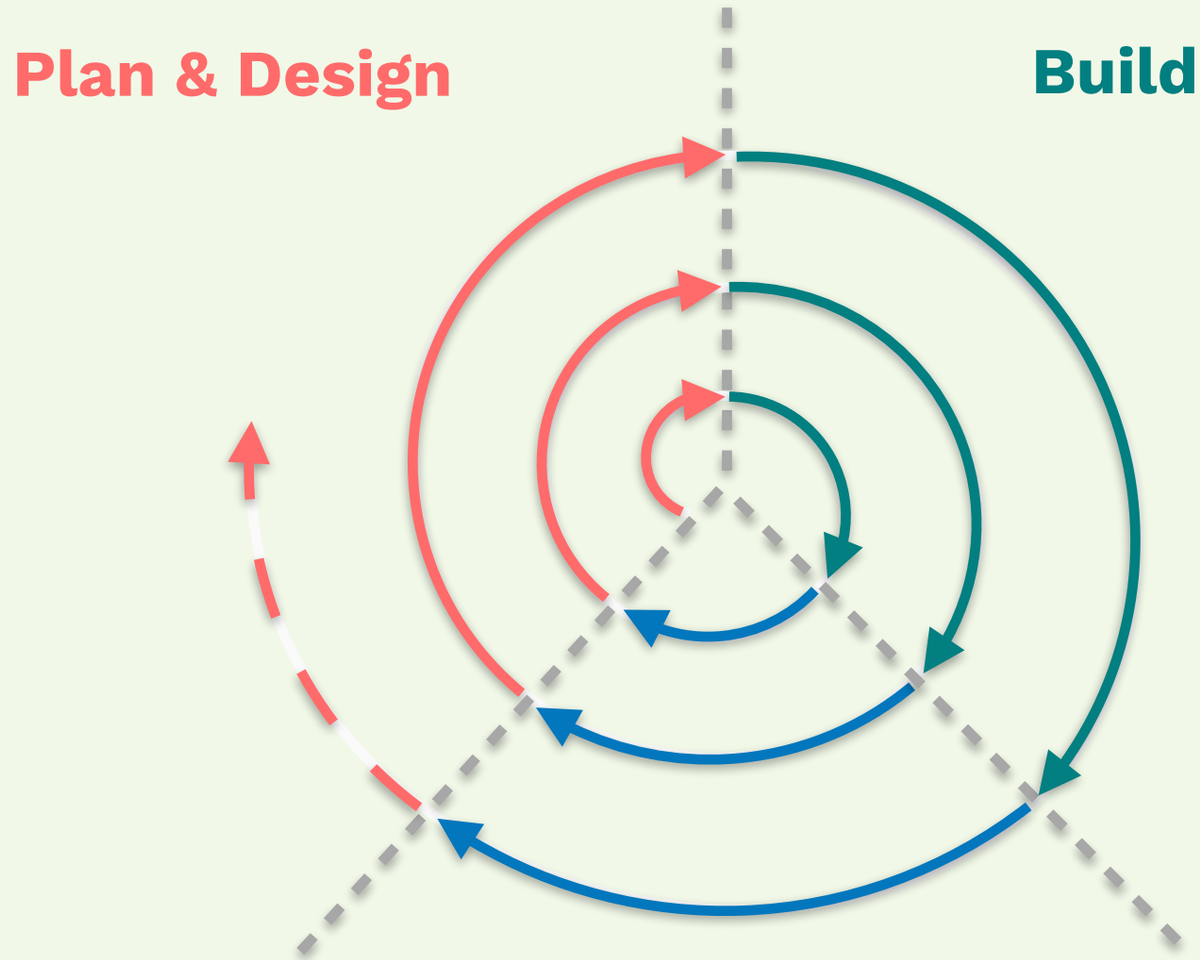
Niall Burkley

@niallburkley

Meltwater

Meltwater API

Iterative Development

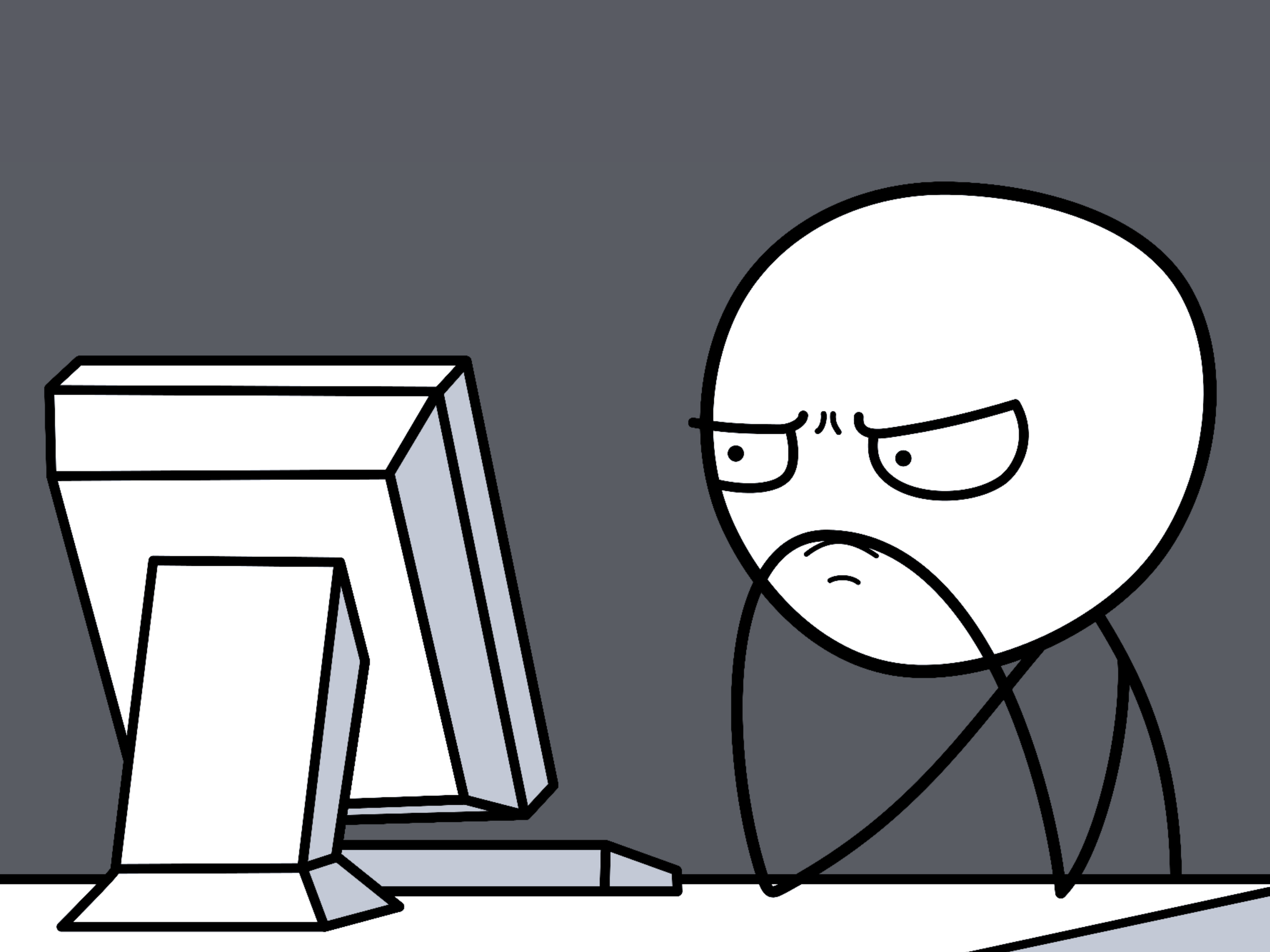


But...

we're building an API?

An API is a contract

Changing our API...



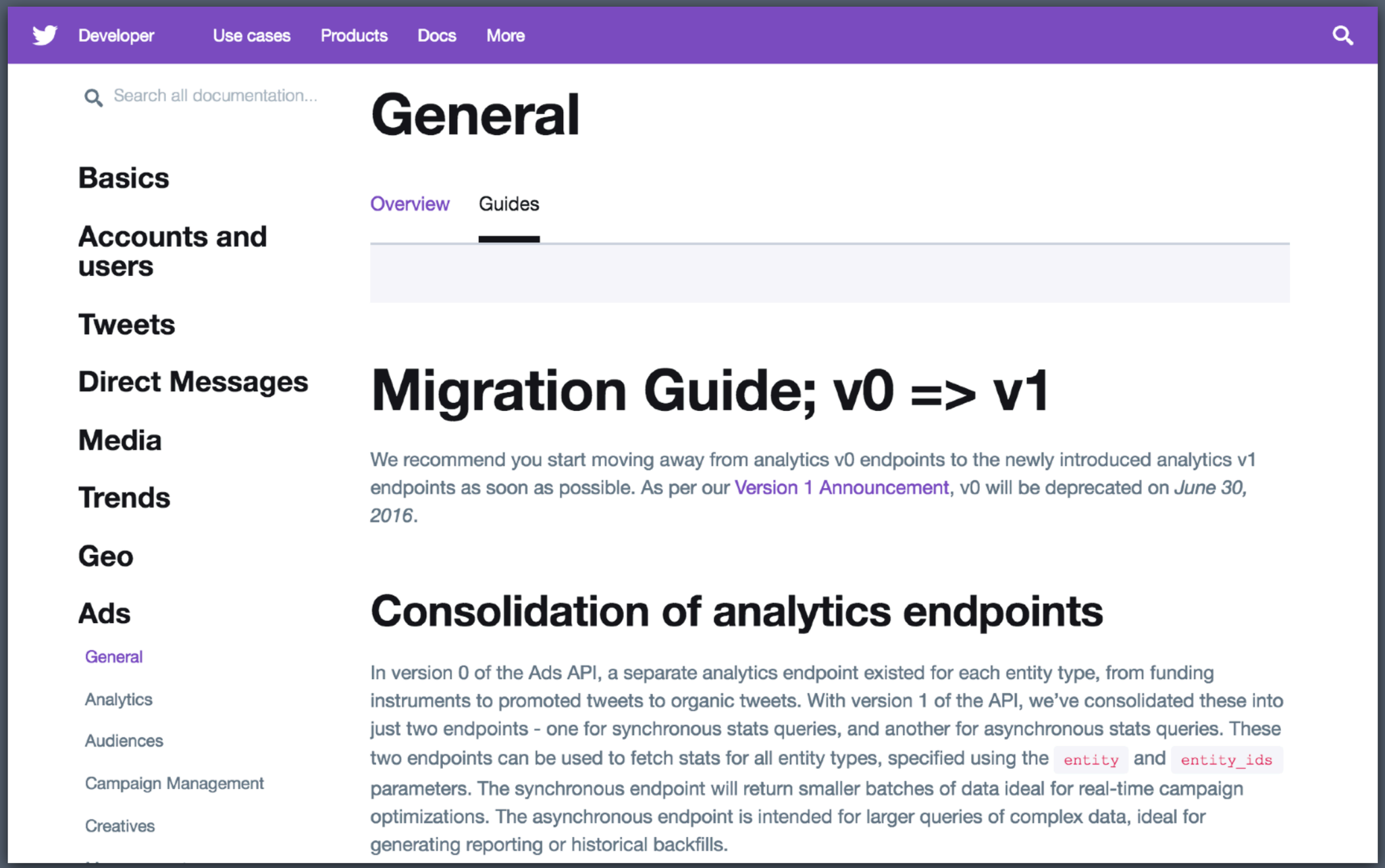
Versioning

/api/**v1**/documents

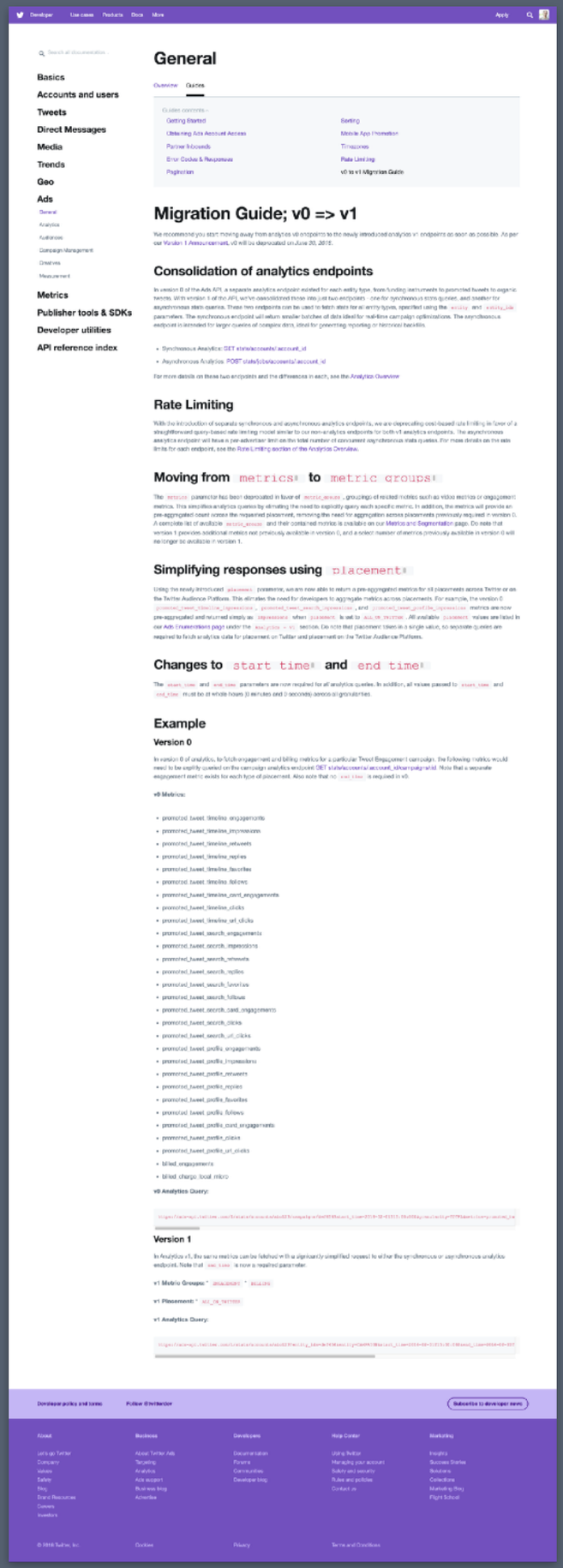
/api/**v2**/documents

/api/**v3**/documents

Migrations


Developer
Use cases
Products
Docs
More
Search all documentation...
Basics
Accounts and users
Tweets
Direct Messages
Media
Trends
Geo
Ads
General
Analytics
Audiences
Campaign Management
Creatives
General
Overview
Guides
Migration Guide; v0 => v1
We recommend you start moving away from analytics v0 endpoints to the newly introduced analytics v1 endpoints as soon as possible. As per our Version 1 Announcement, v0 will be deprecated on June 30, 2016.
Consolidation of analytics endpoints
In version 0 of the Ads API, a separate analytics endpoint existed for each entity type, from funding instruments to promoted tweets to organic tweets. With version 1 of the API, we've consolidated these into just two endpoints - one for synchronous stats queries, and another for asynchronous stats queries. These two endpoints can be used to fetch stats for all entity types, specified using the entity and entity_ids parameters. The synchronous endpoint will return smaller batches of data ideal for real-time campaign optimizations. The asynchronous endpoint is intended for larger queries of complex data, ideal for generating reporting or historical backfills.

Complicated migrations



Developer Use cases Products Docs More Apply

Search all documentation

Basics
Accounts and users
Tweets
Direct Messages
Media
Trends
Geo
Ads
General
Analytics
Audiences
Campaign Management
Creatives
Measurement
Metrics
Publisher tools & SDKs
Developer utilities
API reference index

General

Overview Guides

Guides contents –

Getting Started
Utilizing Ads Account Access
Partner Inbounds
Error Codes & Responses
Pagination
Sorting
Mobile App Promotion
Timezones
Rate Limiting
v0 to v1 Migration Guide

Migration Guide; v0 => v1

We recommend you start moving away from analytics v0 endpoints to the newly introduced analytics v1 endpoints as soon as possible. As per our Version 1 Announcement, v0 will be deprecated on June 20, 2018.

Consolidation of analytics endpoints

In version 0 of the Ads API, a separate analytics endpoint existed for each entity type, from funding instruments to promoted tweets to organic tweets. With version 1 of the API, we've consolidated those into just two endpoints - one for synchronous stats queries, and another for asynchronous stats queries. These two endpoints can be used to fetch stats for all entity types, specified using the `entity` and `entity_ids` parameters. The synchronous endpoint will return smaller batches of data ideal for real-time campaign optimizations. The asynchronous endpoint is intended for larger queries of complete data, ideal for presenting reporting or historical backfills.

- Synchronous Analytics: GET stats/accounts/:account_id
- Asynchronous Analytics: POST stats/jobs/accounts/:account_id

For more details on these two endpoints and the differences in each, see the Analytics Overview

Rate Limiting

With the introduction of separate synchronous and asynchronous analytics endpoints, we are deprecating cost-based rate limiting in favor of a straightforward query-based rate limiting model similar to our non-analytics endpoints for both v1 analytics endpoints. The asynchronous analytics endpoint will have a per-advertiser limit on the total number of concurrent asynchronous stats queries. For more details on the rate limits for each endpoint, see the Rate Limiting section of the Analytics Overview.

Moving from `metrics` to `metric_groups`

The `metrics` parameter has been deprecated in favor of `metric_groups`, grouping of related metrics such as video metrics or engagement metrics. This simplifies analytics queries by eliminating the need to explicitly query each specific metric. In addition, the metrics will provide an pre-aggregated count across the requested placement, removing the need for aggregation across placements previously required in version 0. A complete list of available `metric_groups` and their contained metrics is available on our Metrics and Segmentation page. Do note that version 1 provides additional metrics not previously available in version 0, and a select number of metrics previously available in version 0 will no longer be available in version 1.

Simplifying responses using `placement`

Using the newly introduced `placement` parameter, we are now able to return a pre-aggregated metrics for all placements across Twitter or on the Twitter Audience Platform. This eliminates the need for developers to aggregate metrics across placements. For example, the version 0 `promoted_tweet_timeline_impressions`, `promoted_tweet_search_impressions`, and `promoted_tweet_profile_impressions` metrics are now pre-aggregated and returned simply as `impressions` when `placement` is set to `ALL_ON_TWITTER`. All available `placement` values are listed in our Ads Enumerations page under the `analytics - v1` section. Do note that placement views in a single value, so separate queries are required to fetch analytics data for placement on Twitter and placement on the Twitter Audience Platform.

Changes to `start_time` and `end_time`

The `start_time` and `end_time` parameters are now required for all analytics queries. In addition, all values passed to `start_time` and `end_time` must be at whole hours (0 minutes and 0 seconds) across all granularities.

Example

Version 0

In version 0 of analytics, to fetch engagement and billing metrics for a particular Tweet Engagement campaign, the following metrics would need to be explicitly queried on the campaign analytics endpoint GET stats/accounts/:account_id/campaigns/:id. Note that a separate engagement metric exists for each type of placement. Also note that `no_segmentation` is required in v0.

v0 Metrics:

- promoted_tweet_timeline_engagements
- promoted_tweet_timeline_impressions
- promoted_tweet_timeline_retweets
- promoted_tweet_timeline_replies
- promoted_tweet_timeline_favorites
- promoted_tweet_timeline_follows
- promoted_tweet_timeline_card_engagements
- promoted_tweet_timeline_clicks
- promoted_tweet_timeline_url_clicks
- promoted_tweet_search_engagements
- promoted_tweet_search_impressions
- promoted_tweet_search_retweets
- promoted_tweet_search_replies
- promoted_tweet_search_favorites
- promoted_tweet_search_follows
- promoted_tweet_search_card_engagements
- promoted_tweet_search_clicks
- promoted_tweet_search_url_clicks
- promoted_tweet_profile_engagements
- promoted_tweet_profile_impressions
- promoted_tweet_profile_retweets
- promoted_tweet_profile_replies
- promoted_tweet_profile_favorites
- promoted_tweet_profile_follows
- promoted_tweet_profile_card_engagements
- promoted_tweet_profile_clicks
- promoted_tweet_profile_url_clicks
- billed_engagements
- billed_charge_local_micro

v0 Analytics Query:

Version 1

In Analytics v1, the same metrics can be fetched with a significantly simplified request to either the synchronous or asynchronous analytics endpoint. Note that `end_time` is now a required parameter.

v1 Metric Groups: " " "

v1 Placement: "

v1 Analytics Query:

Developer policy and terms Follow @twitterdev Subscribe to developer news

About
Business
Developers
Help Center
Marketing

© 2018 Twitter, Inc. Cookies Privacy Terms and Conditions

An alternative?

What the user wants...

Stable API

What the user wants...

Documentation

What the user wants...

Easy to Upgrade

What the user wants...

Don’t make me do it

Best for us?

What we want...

Easy to Change

What we want...

Easy to Maintain

What we want...

Incentive to Upgrade

Support ALL the versions!

What!?

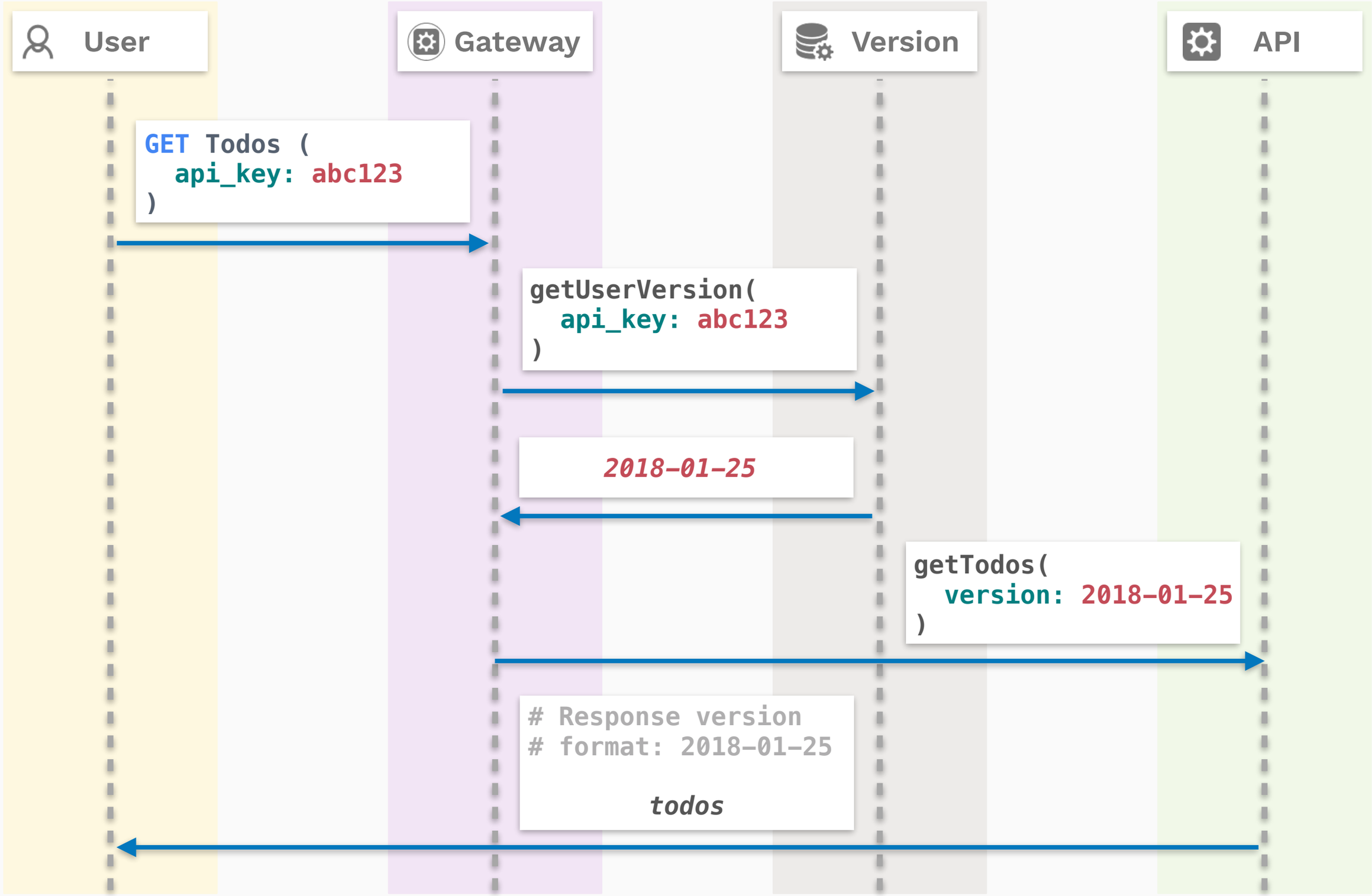
APIs as infrastructure: future-proofing Stripe with versioning

Brandur Leach on August 15, 2017 in Engineering

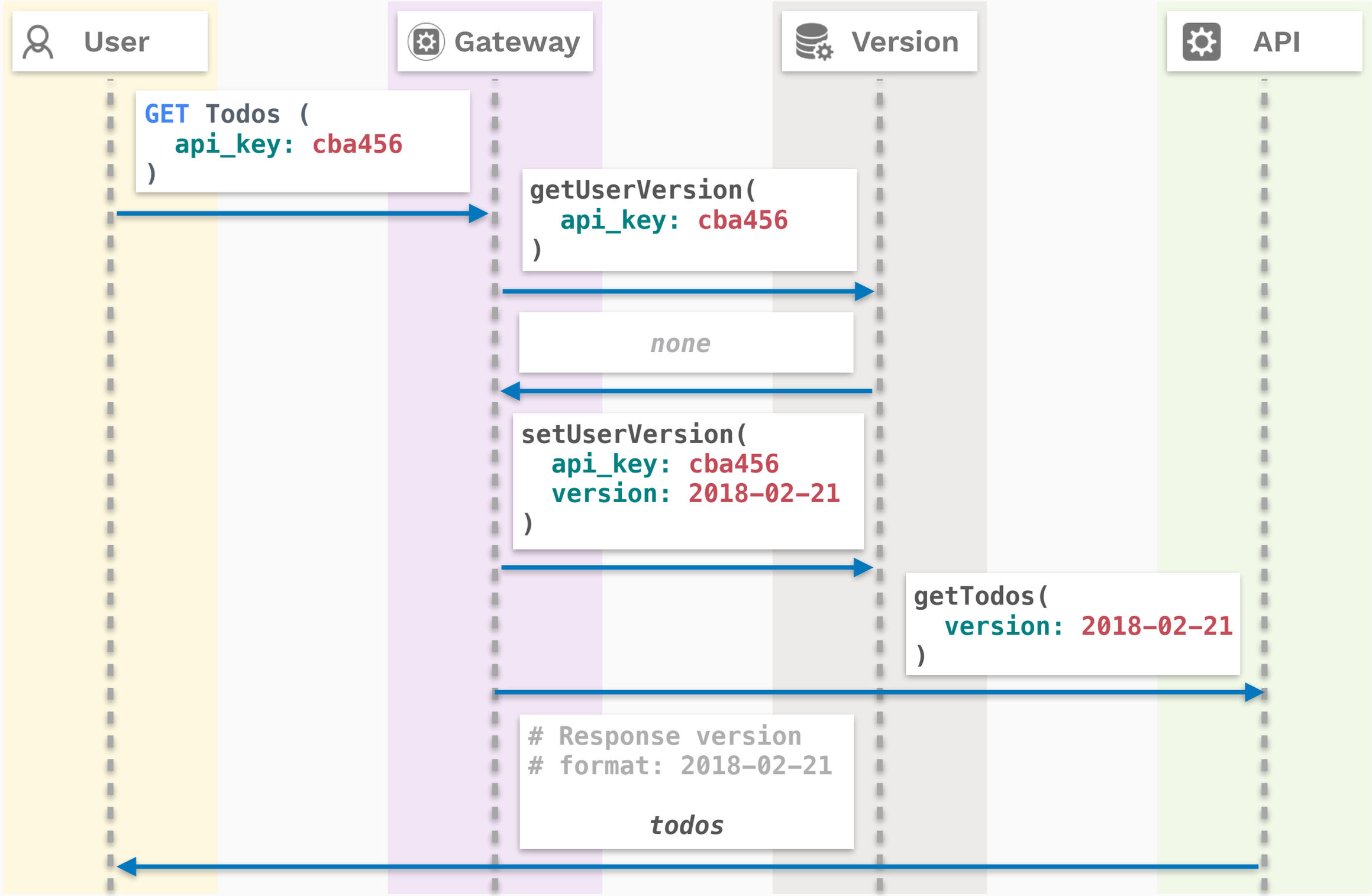
When it comes to APIs, change isn't popular. While software developers are used to iterating quickly and often, API developers lose that flexibility as soon as even one user starts consuming their interface. Many of us are familiar with how the Unix operating system evolved. In 1994, *The Unix-Haters Handbook* was published containing a long list of missives about the software—everything from overly-cryptic command names that were optimized for Teletype machines, to irreversible file deletion, to unintuitive programs with far too many options. Over twenty years later, an overwhelming majority of these complaints are still valid even across the dozens of modern derivatives. Unix had become so widely used that changing its behavior would have challenging implications. For better or worse, it established a contract with its users that defined how Unix interfaces behave.

How does this work for the user?

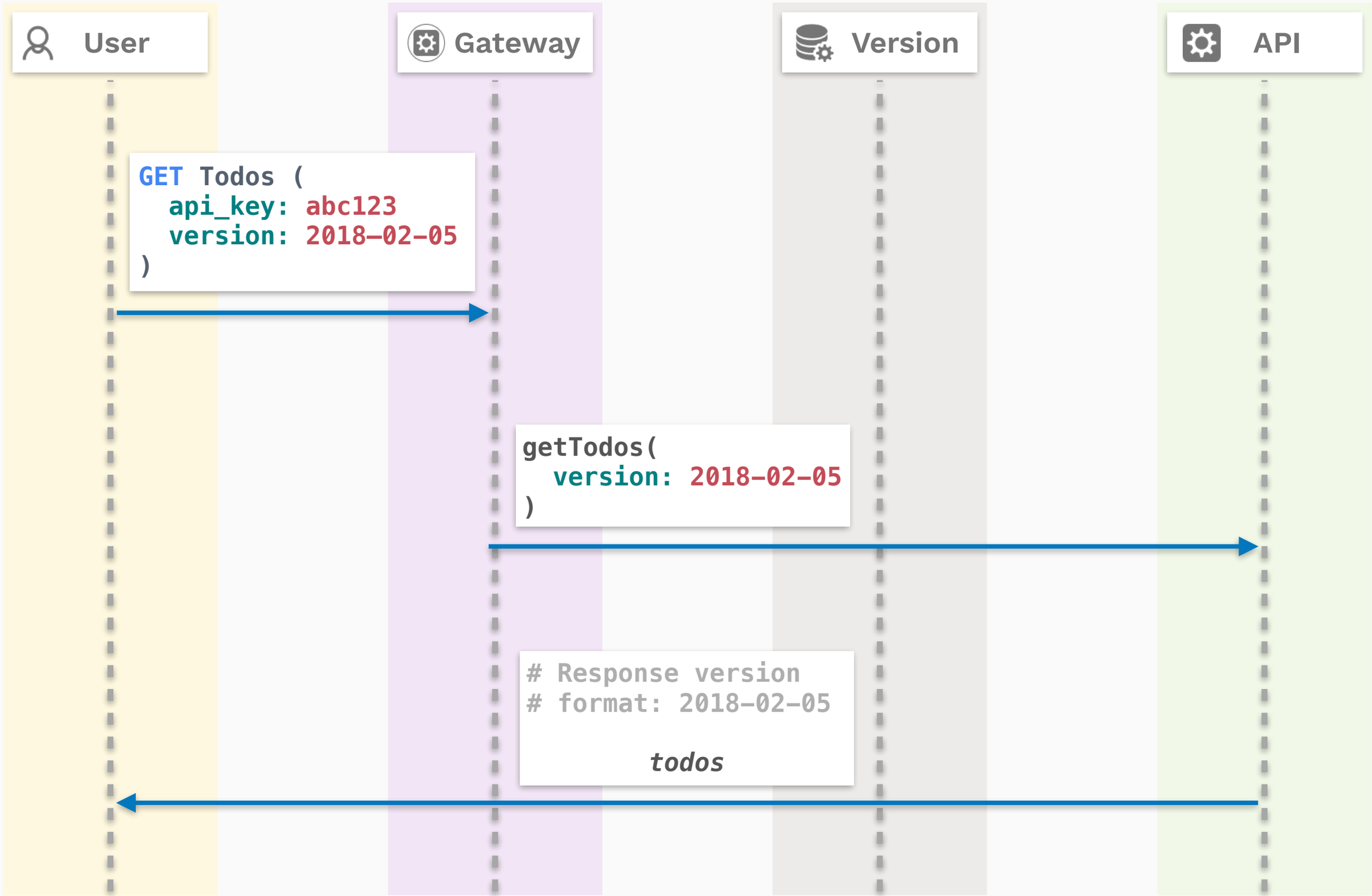
Existing User's API Call



User's first API Call

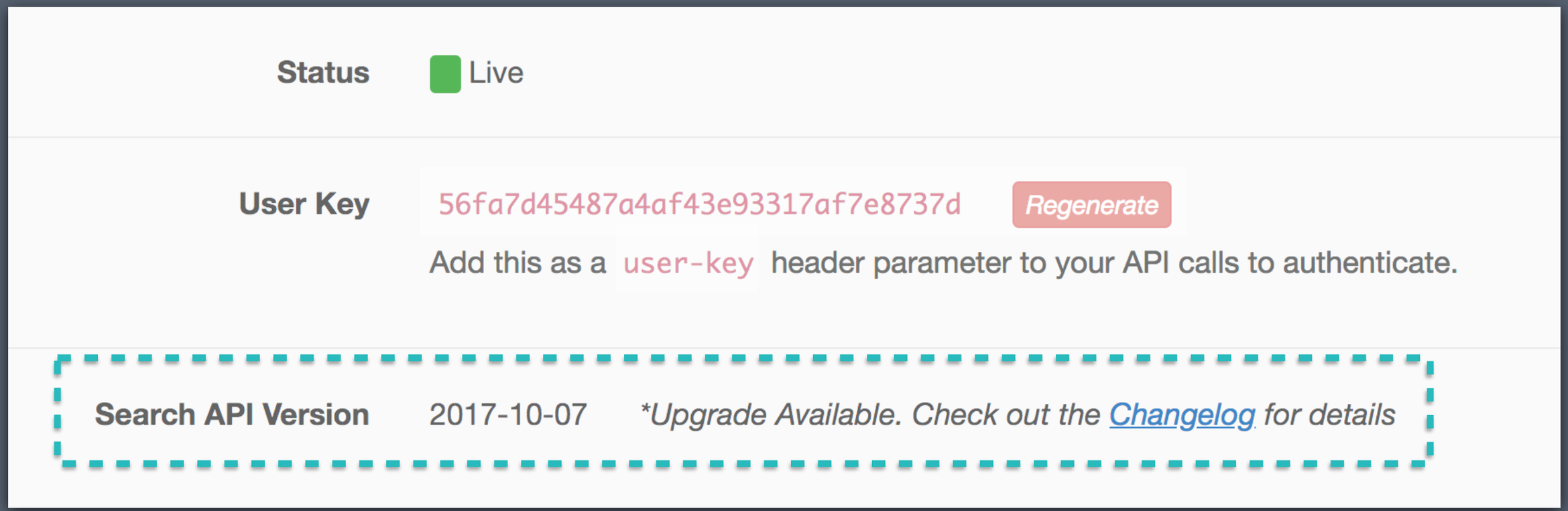


Specifying a different API version

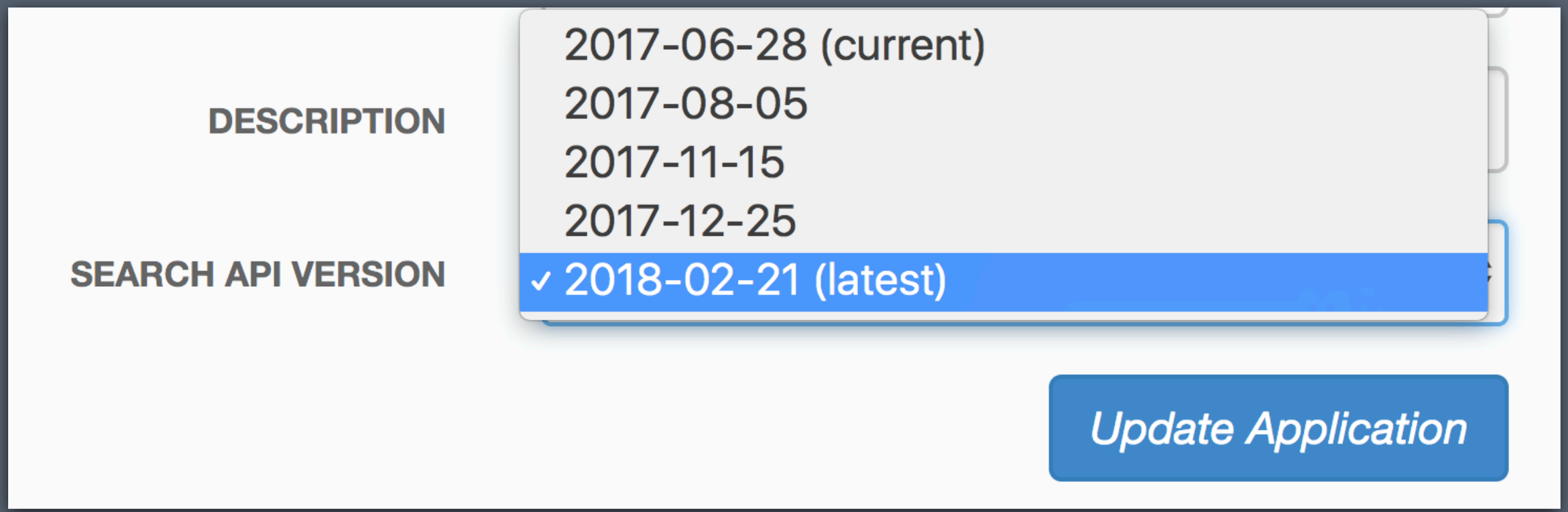


Managing Versions

Managing Versions



Managing Versions



How to build it?

Our implementation...

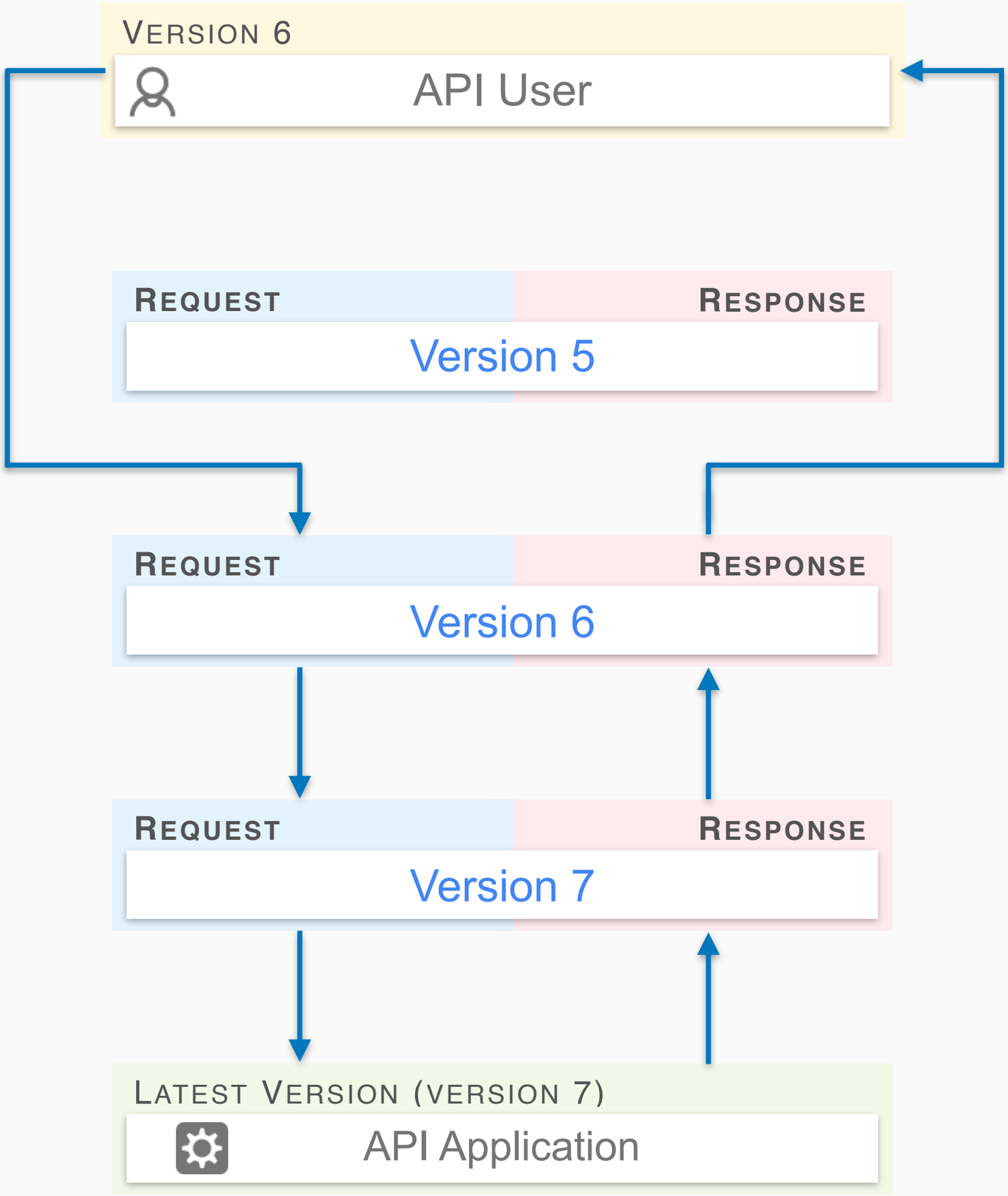
Keep it current

Our implementation...

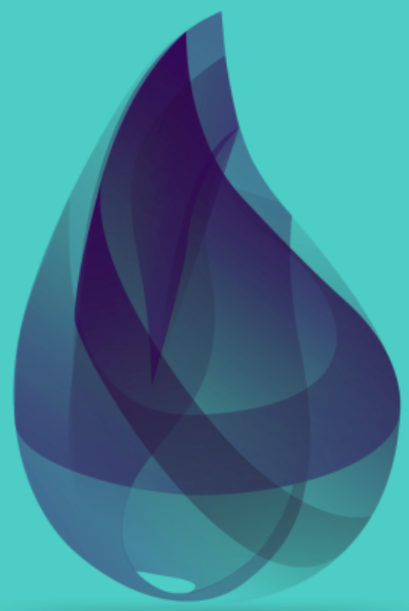
Release rolling versions

Our implementation...

Versions as transformations


VERSION 6
API User
REQUEST
RESPONSE
Version 5
REQUEST
RESPONSE
Version 6
REQUEST
RESPONSE
Version 7
LATEST VERSION (VERSION 7)
API Application

Implementation




TM

```
connection
|> endpoint
|>  router
|> pipeline
|> controller
```

```
connection
|> endpoint
|> plug
|>  router
|> plug
|> pipeline
|> controller
```

```
connection
|> endpoint
|> authentication
|> router
|> apply_version
|> pipeline
|> controller
```

Sample application

```
$> curl -X GET -G /api/todos --data 'size=2'
```

```
$> curl -X GET -G /api/todos --data 'size=2'
```


```
{
  "data": [
    {
      "title": "Build Sample App",
      "id": 1,
      "description": "Put together a sample app for versioning"
    },
    {
      "title": "Add documentation",
      "id": 2,
      "description": "Write up some documentation"
    }
  ]
}
```

```
curl -X GET -G /api/todos --data 'size=2'
```

```
curl -X GET -G /api/todos --data 'page_size=2'
```

Old Version Version

API User

Old Request Format

```
%Plug.Conn{params: %{"size" => size}}
```

Transform Request

Current Request

```
%Plug.Conn{params: %{"page_size" => size}}
```

Latest Version

API Application

```elixir
defmodule TodosWeb.Plugs.ModifyRequest do
  @behaviour Plug

  def init(opts), do: opts

  def call(%Plug.Conn{params: %{"size"=> size} = params} = conn, _) do
    updated_params =
      params
      |> Map.put("page_size", size)
      |> Map.delete("size")

    %{conn | params: updated_params }
  end
  def call(conn, _), do: conn

end
```

```
$> curl -X GET /api/todos
```


```
{
  "data": [
    {
      "title": "Build Sample App",
      "id": 1,
      "description": "Put together a sample app for versioning"
    },
    {
      "title": "Add documentation",
      "id": 2,
      "description": "Write up some documentation"
    }
  ]
}
```

```
...
  {
      {
        "title": "Add documentation",
        "id": 2,
        "description": "Write up documentation"
      }
  }
...
```

```
...
  {
      {
        "title": "Add documentation",
        "id": 2,
        "details": "Write up documentation"
      }
  }
...
```

Latest Version

API Application

Current Response

```
{
  "title": "Add documentation",
  "id": 2,
  “details”: "Write up documentation"
}
```

Transform Response

Old Response Format

```
{
  "title": "Add documentation",
  "id": 2,
  “description”: "Write up documentation"
}
```

Old Version Version

API User

```elixir
defmodule TodosWeb.Plugs.TransformResponse do
  @behaviour Plug

  def init(opts), do: opts

  def call(%Plug.Conn{resp_body: body} = conn, _opts) do
    Plug.Conn.register_before_send(conn, fn conn ->
      transform_description(conn)
    end)
  end
  def call(conn, _), do: conn

  defp transform_description(%Plug.Conn{resp_body: body} = conn) do
  end
end
```

```elixir
defmodule TodosWeb.Plugs.TransformResponse do
  @behaviour Plug

  def init(opts), do: opts

  def call(%Plug.Conn{resp_body: body} = conn, _opts) do
    Plug.Conn.register_before_send(conn, fn conn ->
      transform_description(conn)
    end)
  end
  def call(conn, _), do: conn

  defp transform_description(%Plug.Conn{resp_body: body} = conn) do
  end
end
```

```elixir
defp transform_description(%Plug.Conn{resp_body: body} = conn) do
  json_body = Poison.decode!(body)

  transformed_data =
    json_body["data"]
    |> Enum.map(fn(item) ->
    |> Map.put("details", item["description"])
    |> Map.delete("description")
  end)

  %{conn | resp_body: Poison.encode!(%{json_body | "data" => transformed_data})}
end
```

```elixir
defp transform_description(%Plug.Conn{resp_body: body} = conn) do
  transformed_body =
    body
     |> to_string
     |> String.replace("\"description\":", "\"details\":")

  %{conn | resp_body: transformed_body }
end
```

OLD VERSION VERSION

API User

OLD REQUEST FORMAT

```
%Plug.Conn{params: %{“size” => size}}
```

OLD RESPONSE FORMAT

```
%Plug.Conn{resp_body: body}
```

CURRENT REQUEST

```
%Plug.Conn{params: %{“page_size” => size}}
```

CURRENT RESPONSE

```
%Plug.Conn{resp_body: body}
```

LATEST VERSION

API Application

```elixir
defmodule TodosWeb.Change do

  @doc """
  Transforms the request on the way into the application.
  """
  @callback transform_request(Plug.Conn.t) :: Plug.Conn.t

  @doc """
  Registers callback to transform response on the way out
  of the application
  """
  @callback transform_response(Plug.Conn.t) :: Plug.Conn.t

end
```

```elixir
defmodule TodosWeb.Changes.Versions do

  ...

  @all_versions %{
    "2017-10-02" => [
      TodosWeb.Changes.RevertMultipleAuthors
    ],
    "2017-10-03" => [
      TodosWeb.Changes.RemoveDocumentLocation,
      TodosWeb.Changes.RenameSourceId
    ],
    "2017-10-04" => [
      TodosWeb.Changes.ResetSourceReachDefault
    ]
  }

  ...

end
```

```
defmodule TodosWeb.Changes.Versions do

  ...

  def changes_for(requested_version) do
    @all_versions
    |> versions_since(requested_version)
    |> Keyword.values
    |> List.flatten
  end

  defp versions_since(versions, requested_version) do
    Enum.filter(versions, fn({version_date, _changes}) ->
      requested_version <= version_date
    end)
  end

  ...

end
```

```
connection
 |> endpoint
 |> authentication
 |> router
 |> apply_version
 |> pipeline
 |> controller
```

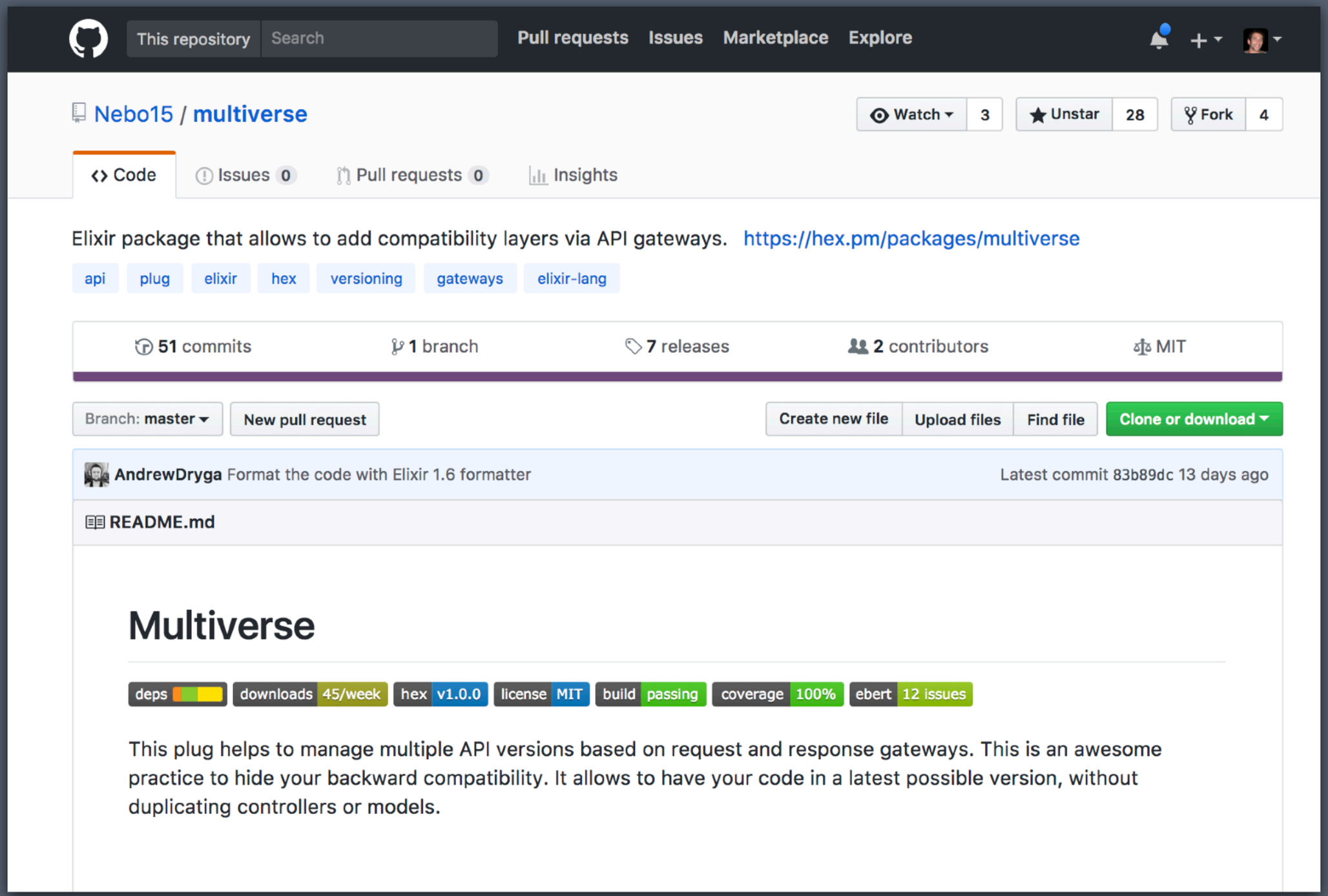
```elixir
defmodule TodosWeb.Plugs.ApplyVersion do
  @behaviour Plug

  def init(opts), do: opts

  def call(conn, _) do
    # 1. get request version

    # 2. get changes for version

    # 3. apply request changes

    # 4. apply response changes

  end
end
```

```elixir
defmodule TodosWeb.Plugs.ApplyVersion do
  @behaviour Plug

  def init(opts), do: opts

  def call(conn, _) do
    changes =
      get_req_header(conn, "x-api-version")
      |> List.first()
      |> TodosWeb.Versions.changes_for()

    # apply request changes
    Enum.reduce(changes, conn, fn change, conn ->
      change.transform_request(conn)
    end)

    # apply response changes
    Enum.reduce(changes, conn, fn change, conn ->
      Plug.Conn.register_before_send(conn, fn conn ->
        change.transform_response(conn)
      end)
    end)

  end
end
```

```elixir
defmodule TodosWeb.Plugs.ApplyVersion do
  @behaviour Plug

  def init(opts), do: opts

  def call(conn, _) do
    changes =
      get_req_header(conn, "x-api-version")
      |> List.first()
      |> TodosWeb.Versions.changes_for()

    # apply request changes
    Enum.reduce(changes, conn, fn change, conn ->
      change.transform_request(conn)
    end)

    # apply response changes
    Enum.reduce(changes, conn, fn change, conn ->
      Plug.Conn.register_before_send(conn, fn conn ->
        change.transform_response(conn)
      end)
    end)

  end
end
```

```elixir
defmodule TodosWeb.Plugs.ApplyVersion do
  @behaviour Plug

  def init(opts), do: opts

  def call(conn, _) do
    changes =
      get_req_header(conn, "x-api-version")
      |> List.first()
      |> TodosWeb.Versions.changes_for()

    # apply request changes
    Enum.reduce(changes, conn, fn change, conn ->
      change.transform_request(conn)
    end)

    # apply response changes
    Enum.reduce(changes, conn, fn change, conn ->
      Plug.Conn.register_before_send(conn, fn conn ->
        change.transform_response(conn)
      end)
    end)

  end
end
```

Multiverse


This repository
Search
Pull requests
Issues
Marketplace
Explore
Nebo15 / multiverse
Watch 3
Unstar 28
Fork 4
Code
Issues 0
Pull requests 0
Insights
Elixir package that allows to add compatibility layers via API gateways. https://hex.pm/packages/multiverse
api
plug
elixir
hex
versioning
gateways
elixir-lang
51 commits
1 branch
7 releases
2 contributors
MIT
Branch: master
New pull request
Create new file
Upload files
Find file
Clone or download
AndrewDryga Format the code with Elixir 1.6 formatter
Latest commit 83b89dc 13 days ago
README.md
Multiverse
deps
downloads 45/week
hex v1.0.0
license MIT
build passing
coverage 100%
ebert 12 issues
This plug helps to manage multiple API versions based on request and response gateways. This is an awesome practice to hide your backward compatibility. It allows to have your code in a latest possible version, without duplicating controllers or models.

Okay…
what about other kinds of changes?

Route Change Between Versions

```
$> curl -X GET /api/todos
```

Route Change Between Versions

```
$> curl -X GET /api/todo_items
```

Route Change Between Versions

```
TodosWeb.Change.TodosRoute do
  @behaviour TodosWeb.Change

  def handle_request(%Plug.Conn{request_path: "/api/todos"} = conn) do
     %{conn | path_info: ["api", "todos_items"] }
  end
  def handle_request(%Plug.Conn{} = conn), do: conn

  def handle_response(%Plug.Conn{} = conn), do: conn
end
```

New Route for Latest Version

New version:

```
curl -X POST /api/todos/share

=> HTTP 200
```

Old version:

```
curl -X POST /api/todos/share

=> HTTP 404
```

New Route for Latest Version

```
TodosWeb.Change.NewShareRoute do
  @behaviour TodosWeb.Change

  def handle_request(%Plug.Conn{request_path: “/api/todos/share”} = conn) do
     # this will internally redirect to a path that doesn't exist
     # The user will get a 404, but since we haven't changed the
     # `request_path` the 404 message will still show /api/todos/share
     %{conn | path_info: ["api", "todos", "NOT_FOUND"] }
  end
  def handle_request(%Plug.Conn{} = conn), do: conn

  def handle_response(%Plug.Conn{} = conn), do: conn
end
```

Route is Removed

```
$> curl -X POST /api/todos/follow
```

Route is Removed

```
$> curl -X POST /api/todos/follow
```

Remove Field from Response Payload

```
{
    "title": "Build Sample App",
    "id": 1,
    "description": "Put together a sample app",
    "tags": ["conference", "code"]
}
```

Remove Field from Response Payload

```
{
    "title": "Build Sample App",
    "id": 1,
    "description": "Put together a sample app",
    "tags": ["conference", "code"]
}
```

Remove Field from Response Payload

```
TodosWeb.Change.RemoveTags do
  @behaviour TodosWeb.Change

  def handle_request(%Plug.Conn{} = conn), do: conn

  def handle_response(%Plug.Conn{request_path: "/api/todos", resp_body: body} = conn) do
    json_body = Poison.decode!(body)

    transformed_data =
      json_body[“data"]
      |> Enum.map(fn(todo_item) -> Map.put(todo_item, ”tags", []) end)

    %{conn | resp_body: Poison.encode!(%{json_body | "data" => transformed_data})}
  end

end
```

Validation Rule Changes

```
{
    "title": "Build Sample App",
    "id": 1,
    "description": "Put together a sample app",
    "tags": ["conference", "code"]
}
```

Max Length: 50

Validation Rule Changes

```
{
    "title": "Build Sample App",
    "id": 1,
    "description": "Put together a sample app",
    "tags": ["conference", "code"]
}
```

Max Length: 30

Validation Rule Changes

```
TodosWeb.Change.TitleValidaton do
  @behaviour TodosWeb.Change

  def init(opts), do: opts

  def handle_request(%Plug.Conn{params: %{"todo"=> %{}} = params} = conn, _) do
    updated_params =
      params
      |> put_in(params, ["todo", "version"], current_version())

    %{conn | params: updated_params }
  end
  def handle_request(conn, _), do: conn

  def handle_response(conn, _), do: conn

end
```

Validation Rule Changes

```
defmodule TodosWeb.Changes.Versions do

  ...

  def max_title_length(version) when version >= "2017-10-03", do: 30

  def max_title_length(_version), do: 50

  ...

end
```

Validation Rule Changes

```
defmodule Todos.TodoList.Todo do
  import Ecto.Changeset

  def changeset(%Todo{} = todo, %{version: version} = attrs) do
    todo
    |> cast(attrs, [:title, :description])
    |> validate_required([:title, :description])
    |> validate_length(:title, max: max_title_length(version))
  end

  defp max_title_length(version) do
    TodoWeb.Versions.max_title_size(version)
  end
end
```

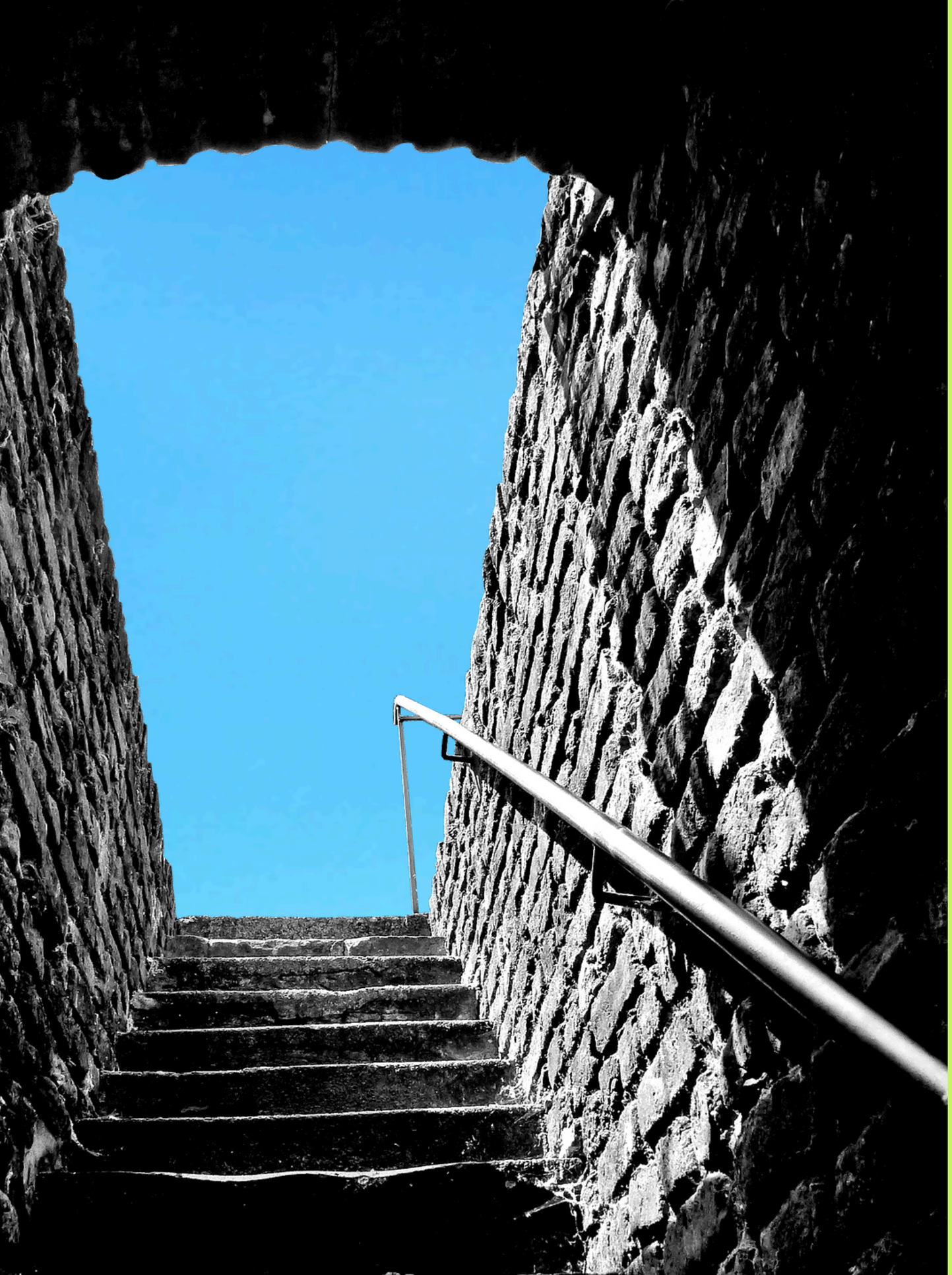
Enough already…

What have we achieved?

Stable API

Documentation

Don’t make me do it



Easy to upgrade

Easy to Change

Incentive to Upgrade

Manageable

Why doesn't everyone do this?

Final Thoughts

@niallburkley | github.com/nburkley | niallburkley.com

Thank you!

Meltwater

underthehood.meltwater.com