The evolution of video APIs

Phil Cluff, July 2019

Me

- Streaming Specialist @ MUX
- Organizer, London Video Tech Meetup 

- Co-Organizer, Demuxed DEMUXED
- Ex VP Engineering @  brightcove®
- Ex Principal Engineer & Team lead @ BBC iPlayer

Coming up

- What are Video APIs?
- Types of Video API
- Why does having a great API matter?
- Considerations when designing a great Video API

MUX

But first...

What makes me qualified to talk about this?

- I’ve built private video APIs, interfacing with biggest SAAS encoding vendors at the BBC
- I’ve built public and private video APIs at Brightcove / Zencoder
- I now build API first video services at Mux

MUX

What are Video APIs?

MUX

My Definition:

- Anything you use to programmatically manipulate video, **is a Video API**
- I'll focus on about public SAAS Video APIs today
- Generally this break down into 2 categories:
  - Encoding APIs
  - Video Platform APIs

MUX

Types of Video API:

Encoding APIs

MUX

Encoding APIs

- “Encoding APIs” is a bad name - a pure SAAS encoder API doesn’t exist.
- “Encoding APIs” are really Transcoders, Muxers, Packagers, File transfer agents, etc.
- Fine grained control over encoding settings
- Tend to work with remote storage - S3, GCS, etc.

MUX

Encoding APIs

Timeline of Encoding API releases

brightcove ZENCODER

- Simple JSON / XML API
- Trivial simplest job
- Temporary output storage
- Some SDKs

Amazon Elastic Transcoder

- AWS Style JSON API
- More complex API
- Pipelines concept separates input / output / auth from job
- Good SDK support

BITMOVIN
VIDEO INFRASTRUCTURE FOR THE WEB

- Originally called “Bitcodin”
- Complex JSON API with large minimum payload
- Good SDK support
- Bitmovin actively encourage you to use SDKs

AWS Elemental MediaConvert

- AWS SAAS version of Elemental Cloud
- Advanced, Broadcaster friendly
- Complex JSON API, accessed via SDKs or CLI, passes huge JSON Job specifications
- Good SDK support

2010	2011	2012	2013	2014	2015	2016	2017	2018	2019	2020

HDS, Smooth, HLS

Smooth, HLS, DASH

CMAF, LHLS, Low latency DASH

MUX

How have encoding APIs evolved?

- Granularity of control has increased dramatically
- Complexity of API exchange has grown dramatically

MUX

Example: Zencoder V2 simplest job

```
POST /api/v2/jobs
{
    “input” : “https://foo.com/video.mp4”
}
```

MUX

Example: Media Convert Simplest Job

```
{
  "Queue": "arn:aws:mediaconvert:us-east-1:209867822108:queues/Default",
  "UserMetadata": {},
  "Role": "arn:aws:iam::209867822108:role/media-convert",
  "Settings": {
    "OutputGroups": [
      {
        "Name": "File Group",
        "Outputs": [
          {
            "ContainerSettings": {
              "Container": "MP4",
              "Mp4Settings": {
                "CslgAtom": "INCLUDE",
                "FreeSpaceBox": "EXCLUDE",
                "MoovPlacement": "PROGRESSIVE_DOWNLOAD"
              }
            },
            "VideoDescription": {
              "Width": 640,
              "ScalingBehavior": "DEFAULT",
              "Height": 480,
              "TimecodeInsertion": "DISABLED",
              "AntiAlias": "ENABLED",
              "Sharpness": 50,
              "CodecSettings": {
                "Codec": “H_264",
               "H264Settings": {
                  "InterlaceMode": "PROGRESSIVE",
                  "NumberReferenceFrames": 3,
                  "Syntax": “DEFAULT",
                  "Softness": 0,
                  "GopClosedCadence": 1,
                  "GopSize": 90,
                  "Slices": 1,
                  "GopBReference": "DISABLED",
                  "SlowPal": "DISABLED",
                  "SpatialAdaptiveQuantization": "ENABLED",
                  "TemporalAdaptiveQuantization": "ENABLED",
                  "FlickerAdaptiveQuantization": "DISABLED",
                  "EntropyEncoding": "CABAC",
                  "Bitrate": 10000,
                  "FramerateControl": "INITIALIZE_FROM_SOURCE",
                  "RateControlMode": "CBR",
                  "CodecProfile": "MAIN",
                  "Telecine": "NONE",
                  "MinIInterval": 0,
                  "AdaptiveQuantization": "HIGH",
                  "CodecLevel": "AUTO",
                  "FieldEncoding": "PAFF",
                  "SceneChangeDetect": "ENABLED",
                  "QualityTuningLevel": "SINGLE_PASS",
                  "FramerateConversionAlgorithm": "DUPLICATE_DROP",
                  "UnregisteredSeiTimecode": "DISABLED",
                  "GopSizeUnits": "FRAMES",
                  "ParControl": "INITIALIZE_FROM_SOURCE",
                  "NumberBFramesBetweenReferenceFrames": 2,
                  "RepeatPps": "DISABLED",
                  "DynamicSubGop": "STATIC"
                }
              },
              "AfdSignaling": "NONE",
              "DropFrameTimecode": "ENABLED",
              "RespondToAfd": "NONE",
              "ColorMetadata": "INSERT"
            },

            "AudioDescriptions": [
              {
                "AudioTypeControl": "FOLLOW_INPUT",
                "CodecSettings": {
                  "Codec": "AAC",
                  "AacSettings": {
                    "AudioDescriptionBroadcasterMix": "NORMAL",
                    "Bitrate": 96000,
                    "RateControlMode": “CBR",
                    "CodecProfile": "LC",
                    "CodingMode": "CODING_MODE_2_0",
                    "RawFormat": "NONE",
                    "SampleRate": 48000,
                    "Specification": "MPEG4"
                  }
                },
                "LanguageCodeControl": "FOLLOW_INPUT"
              }
            ],
            "Extension": "mp4",
            "NameModifier": "foo"
          }
        ],
        "OutputGroupSettings": {
          "Type": "FILE_GROUP_SETTINGS",
          "FileGroupSettings": {
            "Destination": "s3://phils-players-video/outputs/test"
          }
        }
      }
    ],
    "AdAvailOffset": 0,
    "Inputs": [
      {
        "AudioSelectors": {
          "Audio Selector 1": {
            "Offset": 0,
            "DefaultSelection": "DEFAULT",
            "ProgramSelection": 1
          }
        },
        "VideoSelector": {
          "ColorSpace": "FOLLOW",
          "Rotate": "DEGREE_0"
        },
        "FilterEnable": "AUTO",
        "PsiControl": "USE_PSI",
        "FilterStrength": 0,
        "DeblockFilter": "DISABLED",
        "DenoiseFilter": "DISABLED",
        "TimecodeSource": "EMBEDDED",
        "FileInput": "s3://phils-players-video/inputs/big_buck_bunny_1080p_h264.mov"
      }
    ]
  },
  "StatusUpdateInterval": "SECONDS_60"
}
```

MUX

But Why?

- Less active targeting of a developer market?
- More targeting of a high end, systems integration market?
- Less upfront design going into HTTP APIs?

MUX

Types of Video API:

Video Platform APIs

MUX

Video Platform APIs

- Video Platform APIs are those provided by Online Video Platforms (OVPs)
- OVPs tend to be closed ecosystems, providing:
  - Ingest, Transcode, Storage, Catalog, Playback, Distribution, CDN, Analytics etc. all through one product.
- Video Platform APIs tend to be composed of 3 main parts:
  - Catalog
  - Ingest
  - Playback

MUX

Video Platform APIs

MUX

Video Platform APIs: Catalog

- Management of content in your account, mainly metadata about the content. EG:
  - Title
  - Description
  - Categories
  - Tagging

MUX

Video Platform APIs: Ingest

- Exposes some control over the encoding process.
- Some use ingestion “profiles” to define encode behavior
  - “profiles” may have their own API to provide better abstraction
- Others can look like “Encoding APIs”

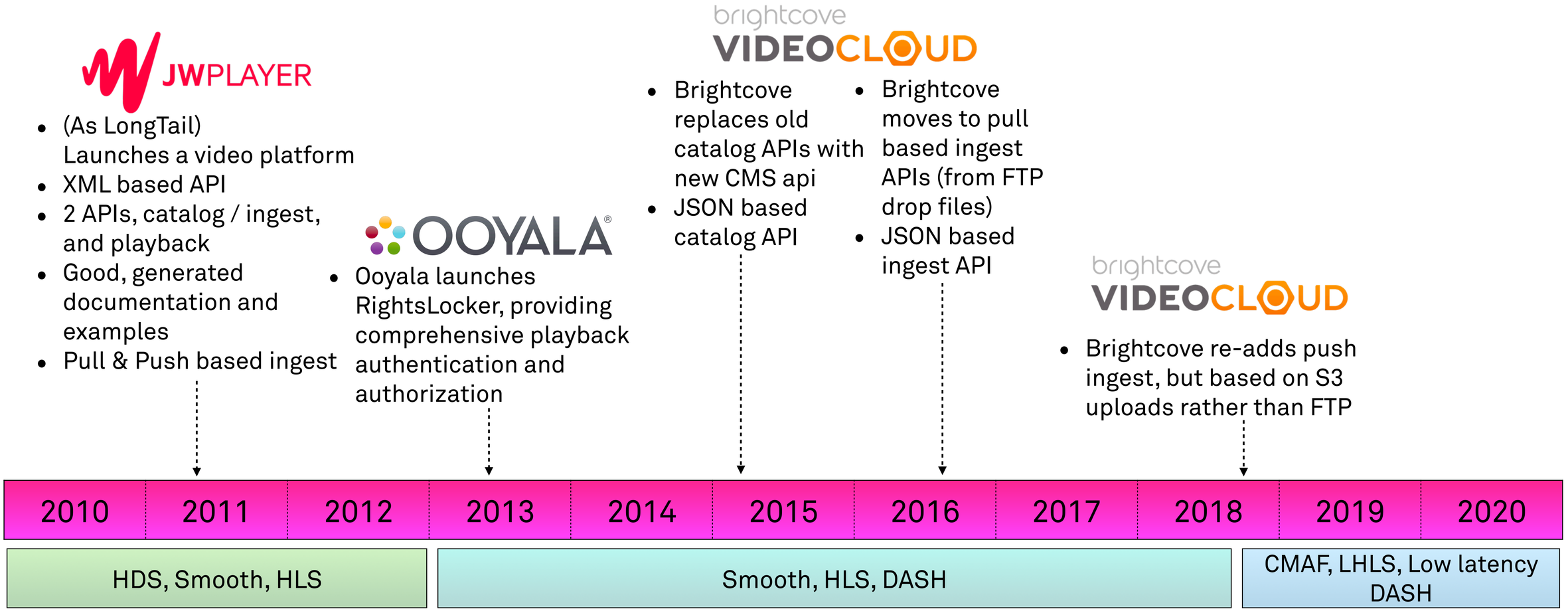
MUX

Video Platform APIs: Playback

- Used to get URLs for the playback of content
  - HLS, DASH, Smooth, pmp4 URLs
  - Manifest URLs may be tokenized with expiring URLs
- Usually secured, but often not secured well

MUX

Timeline of changes to Video Platform APIs



MUX

How have Video Platform APIs evolved?

- Move from “Push” APIs for Ingest to “Pull” APIs like encoders
  - But “Push” APIs are still around, just not FTP anymore
- Emergence of Playback security

MUX

Playback Security is improving

- Traditionally playback APIs are a weak point in OVP API security
- This isn’t DRM, just Authorization and Authentication of Playback APIs
- But this is improving!
- Ooyala’s “Rights Locker” was a great example

MUX

Why haven’t Video Platform APIs evolved more?

- OVPs APIs tend to be slow-moving
- Fear of breaking customer integrations
  - Customer integrations are complex, and not well understood
  - Integrations may have been outsourced
  - Lack of desire to support multiple versions of APIs results in either shoehorning or long migration periods

MUX

Types of Video API:

Other Video APIs

MUX

FFmpeg

- FFmpeg is integrated into most Video products
- Everyone integrates it using the command line interface
- So FFmpeg's CLI is a Video API, right?
  - Not a SAAS API, but we use it to manipulate video...
- We should probably be using libav, but who has time for that?

MUX

FFmpeg has cool defaults...

- What do the following commands do?
  - `ffmpeg -i ~/in.mp4 out.mp4`
  - `ffmpeg -i ~/in.mp4 out.webm`

MUX

MUX's /video API

- Video Infrastructure as a service
- Built with similar API concepts to Zencoder
- Trivial API for ingest and streaming of content
- Somewhere between an encoding API, and a platform API without a catalog

MUX

MUX's /video API

```
POST /video/v1/assets/
{
    "input": "https://foo.com/video.mp4",
    "playback_policy": ["public"] }
}

200 OK
{
    "data" : {
        …
        “playback_ids”:
[{"policy":"public","id":"AZ67d3CrF8dMzMD02rl5024701STWnjoo1A"}]
    }
}
```

MUX

MUX's /video API

HLS:

https://stream.mux.com/AZ67d3CrF8dMzMD02rl5024701STWnjoo1A.m3u8

Thumbnail:

https://image.mux.com/AZ67d3CrF8dMzMD02rl5024701STWnjoo1A/thumbnail.png

Animated Gif:

https://image.mux.com/AZ67d3CrF8dMzMD02rl5024701STWnjoo1A/animated.gif

MUX

Why does API structure matter?

SAAS is the new normal

- Your customers want to be able to experiment with your product without writing hundreds of lines of code or thousands of lines of JSON
- Developers are increasingly empowered to make vendor decisions
- Particularly in a developer centric market

MUX

Developer market == growth market

- There’s a finite number of broadcasters and media companies
- There’s a virtually infinite cycle of new startups who want video experiences
- There’s huge growth opportunities in being friendly to developers who want to do things with video
- You can empower normal full-stack developers with great APIs to build video apps

MUX

Why are we bad at this?

- **Engineers:** We tend to be from a video engineering background, not a SAAS background, we get lost in the weeds of video, we design APIs for video developers, not for generic developers.
- **Product:** Product tends to plan around functionality, and GUI, not API experience

MUX

Considerations when designing a great Video API

MUX

Design

Design: Upfront design your API as part of Product

- Your API design should be a part of your product process, not an afterthought

MUX

Design: Talk to customers about your API design

- Customers want to have a 2 way relationship
- Customers will give you valuable honest feedback about APIs
- Remember: They’re going to be using it, not you

MUX

Design: Consider using API definition toolchains

- There are a lot of toolchains for API design and description
- OpenAPI 3 (previously swagger) is great
  - ^ We use this
- API Blueprint is another option
- RAML upcoming interesting standard

MUX

Abstraction

Abstraction: Think about Abstraction up front

- Ask yourself: (or even better, ask your customers)
  - What problem are you trying to solve?
  - What's your target market?
  - What do my customers know anything about video?
- One level of Abstraction per API is important
  - Re-arrange your APIs if you find your abstraction is bad

MUX

Abstraction: Revisit as your API evolves

- Its easy to let APIs mutate as they evolve
- APIs that grow organically aren’t generally the best APIs
- You should think about your API design every time you make an API change

MUX

Simplicity

MUX

Simplicity: Have sensible default behavior

- Default behavior should be simple, and help customers get started easily
- Some ideas for defaults:
  - Input File => MP4 File? EG: Zencoder
  - Input File => HLS ABR? EG: Mux
  - Default to H.264?
  - Follow output filename? EG: ffmpeg

MUX

Ease of Integration

MUX

Integration: Make the simplest integration trivial

- More than just sensible defaults
- Small minimum API request size
- Trivial authentication
  - Probably not OAuth...
- Ask yourself...

MUX

Could I use this API with just curl?

MUX

Example: OAuth Getting Started...

Quick Start: OAuth

Video Cloud
API Developer
Authentication, Code Samples, OAuth API

This tutorial guides you through the steps to get client-credentials and an access token using the OAuth API. Before starting the tutorial, you should read OAuth Overview.

Requirements

- API calls in this tutorial use cURL, which is built in on MacOS and all linux/unix systems. If you are running Windows, see Set Up cURL
- You will need Terminal or some other command-line console for your system

Overview

There are 2 parts to using the Brightcove OAuth API to get access to other Brightcove APIs

- Get client credentials (a `client_id` and `client_secret`) that are valid for the account(s) and API operation(s) you need - this is a one-time operation, provided you save your client_id and client_secret in some secure place for future use
- Get an `access_token` that provides authorization to make an API call - the access_token is valid for 5 minutes, so in most cases you will get one for each API call you make

There is also a UI in Studio that you can use to obtain and manage client-credentials. It is recommended that you use the UI whenever possible. For new APIs, the credentials may not be available via the UI for some time, in which case you can get credentials directly from the OAuth API, as you will learn how to do in the steps that follow.

Get client credentials

In these steps we will use a cURL command to get client credentials for the Player Management API.

Steps

1. Make sure you have cURL available as explained in Requirements; you can test by going to a command line and typing `curl` and pressing return - if cURL is installed, you should see a message like this:

```
curl: try 'curl --help' or 'curl --manual' for more information
```

2. Now you need two pieces of information that you can obtain from Studio. Login to Studio by going to Studio.
   - account id
   - BC_TOKEN

   Note: The `BC_TOKEN` value is a cookie set when you open Studio. It has no relation to the API Management tokens you deal with in Studio.

Get your account id and BC_TOKEN

3. First you will need your account id from the Account Information.
4. Click the settings icon ⚙ in the Studio header.
5. Select Account Information from the dropdown.
6. Copy your account id from the Account Information page and save it - you will need it in a later step.

7. Now open the browser developer console - in most browsers you can do this by pressing OPTION-CMD-i (Mac) or ALT-CTRL-i (Windows), or you should be able to find a menu item also.
8. In the console, paste the following JavaScript and press return to display your BC_TOKEN in a modal dialog:

```
var cookiesArray = document.cookie.split(";"), cookiesObj = {}, i, tmpArray = [];
for (i = 0; i < cookiesArray.length; i++) {
    tmpArray = cookiesArray[i].split("=");
    if (tmpArray[0].indexOf('BC_TOKEN') > -1) {
        cookiesObj.BC_TOKEN = tmpArray[1];
    }
}
window.prompt("BC_TOKEN:", cookiesObj.BC_TOKEN);
```

9. Copy the BC_TOKEN to the clipboard.
10. Enter your account id and BC_TOKEN here:

Account id/BC_TOKEN

Account id: 876457237

BC_TOKEN:

```
EnTxTg5fValF5Mso-TiOxmUI2Ooa59KJmXMiprB-
Z4SZYumeNGkx6VToR0CvYGxxN0lVKH6BqE2dTF9iPyEJt3BewgJVEih2RFBKjYAtl8thJmPCkI6
```

Make the credentials request

11. To get your client credentials, copy and paste the following cURL command at your command line and press return:

```
Your cURL command will appear here after you enter you account id and BC_TOKEN
```

12. The response should look like this (formatting added):

```
{"redirect_url":null,"maximum_scope":[{"identity":
{"type":"video-cloud-account","account-id":57838016001},
"operations":["video-cloud/player/all"]}],"name_html":"Sample-Client",
"issued_to":"rcrooks@brightcove.com","trusted":null,"expires_at":null,
"issued_at":"2015-06-18T20:17:12Z","name":"Sample-Client",
"description_html":null,"revoked":null,"type":"credential",
"client_secret":"PiQ5s3-tk46DvNBPauknctP3aTI507zNIa7Kxz3b7hnRMA5JvQdkwa7JpyuX7dnL
"description":null,"client_id":"b63e5ac2-5264-4a5a-971a-a133bc7bd605",
"issued_user":53255203001}
```

This is **not** a valid client secret - you should **never** expose a client secret publicly.

13. Enter the values for the `client_id` and `client_secret` below, as you will need these anytime you need to get an `access_token`.

Client Credentials

client_id:

client_secret:

Submit id and BC_TOKEN

Get access tokens

The `client_id` and `client_secret` you obtained in the previous steps are used as a *username:password* combination for authentication when you request an `access_token`. In cURL, which we will use here, you can pass these as the `--user` parameter. In other languages, you will pass these in a Basic Authorization header with your HTTP request:

```
Authorization: Basic {client_id:client_secret}
```

Note: the entire client_id:client_secret *must* be Base64-encoded to be accepted by the OAuth API. cURL will Base64-encode the --user credentials automatically, but in other languages you will need to Base64 encode it yourself.

Steps

14. To get your access_token, copy and paste the following cURL command to your command line and press enter:

```
Your cURL command will appear here after you enter you client_id and client_secret
```

15. The output should look like this:

```
{"access_token":"ACkkpftcuhyzqdf4ftxm304za3anhziG0Ni5-S_dplxMNYNrSrBZRgJL3sf_U38Z
```

16. This token can be used to authenticate calls to Player Management API for 5 minutes (then you need to get a new token). The token is passed in the Authorization header with the HTTP request:

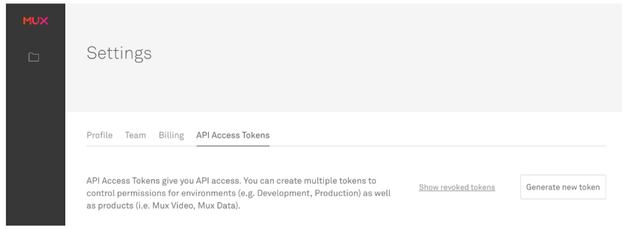
```
Authorization: Bearer access_token value
```

MUX

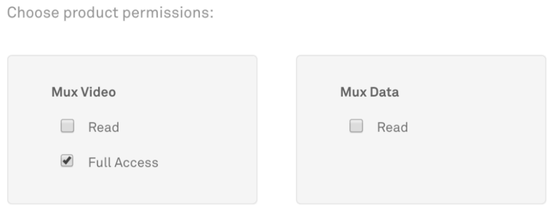
Example: API Token getting started

1. Get an API Access Token

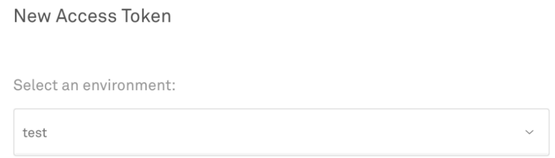
The Mux Video API uses a token key pair that consists of a **Token ID** and **Token Secret** for authentication. If you haven't already, generate a new Access Token in the Access Token settings of your Mux account dashboard.



The Access Token should be set to "Full Access" for Mux Video.



Access Tokens also belong to an Environment. Be sure to use the same Environment when using Mux Video and Mux Data together, so the data from Mux Data can be used to optimize your Mux Video streams.



2. POST a video

(Detailed API Reference)

Videos stored in Mux are called assets. To create your first video asset, send a POST request to the /assets endpoint and set the "input" property to the URL of a video file that's accessible online.

The biggest SAAS vendors use API tokens

- Stripe
  - `GET https://api.stripe.com/v1/charges HTTP/1.1`
    `Authorization: Bearer Your.API.Key-HERE`
- Twillio, SendGrid
  - `GET https://api.sendgrid.com/v3/mail HTTP/1.1`
    `Authorization: Bearer Your.API.Key-HERE`

MUX

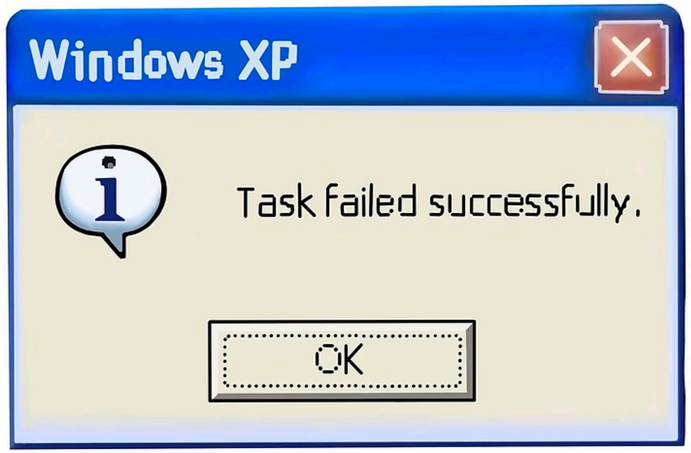
Integration: Document your APIs well

- Customers won’t use your APIs if they aren’t well documented
- Document your APIs in HTTP form as well as via the SDKs
- This can be done programmatically if you’re using API definition files like Open API

MUX

Integration: Have sensible error messages

- Great error messages help API users integrate
- Validate early, use sensible HTTP codes
- Don’t be Windows XP



MUX

Just the right amount of

Stickiness

MUX

Stickiness: Just the right amount

- If your APIs are great, your product will naturally be sticky
- But you want to avoid customers feeling locked in

MUX

Stickiness: Version your APIs

- At some stage you will want to release a new version of your API with breaking changes
- Make sure this isn’t the first time you introduce a version number to your URLs
- Have your API version early in the HTTP path, EG:
  - https://api.mux.com/video/v1/assets

MUX

Stickiness: Provide migration paths

- Forcing customers to migrate between versions of an API isn’t good for customer retention
- You should almost always replace old APIs with shims, which recreate old APIs, using the new APIs
- Use a carrot (New features?) to tempt customers to upgrade to new APIs, rather than a stick (We’re going to turn the old one off!)

MUX

But...

MUX

Be ready to get it wrong!

And iterate!

MUX

Summary

- Generally, there's 2 types of Video APIs, Encoding and Platform
- APIs have got more complex, and more detailed over the last 10 years, perhaps needlessly
- A great API is critical in a self-service SAAS environment, and there's huge opportunity in targeting a developer market
- Actively design your APIs for elegant abstraction, simplicity, easy of integration, and stickiness

MUX

MUX

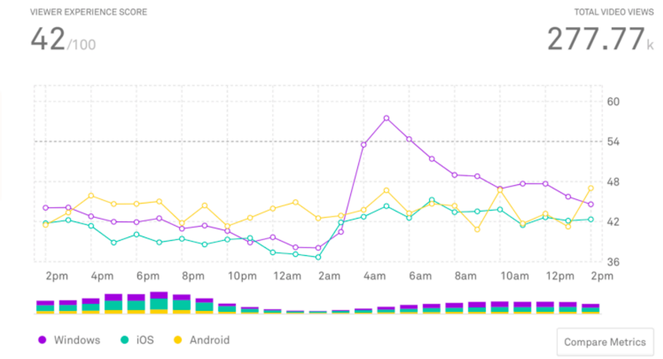
Thank you!

Questions? Corrections?

phil@mux.com

/data

Best-of-breed live and on-demand video streaming analytics.



/video

An API to build amazing video experiences for any team.