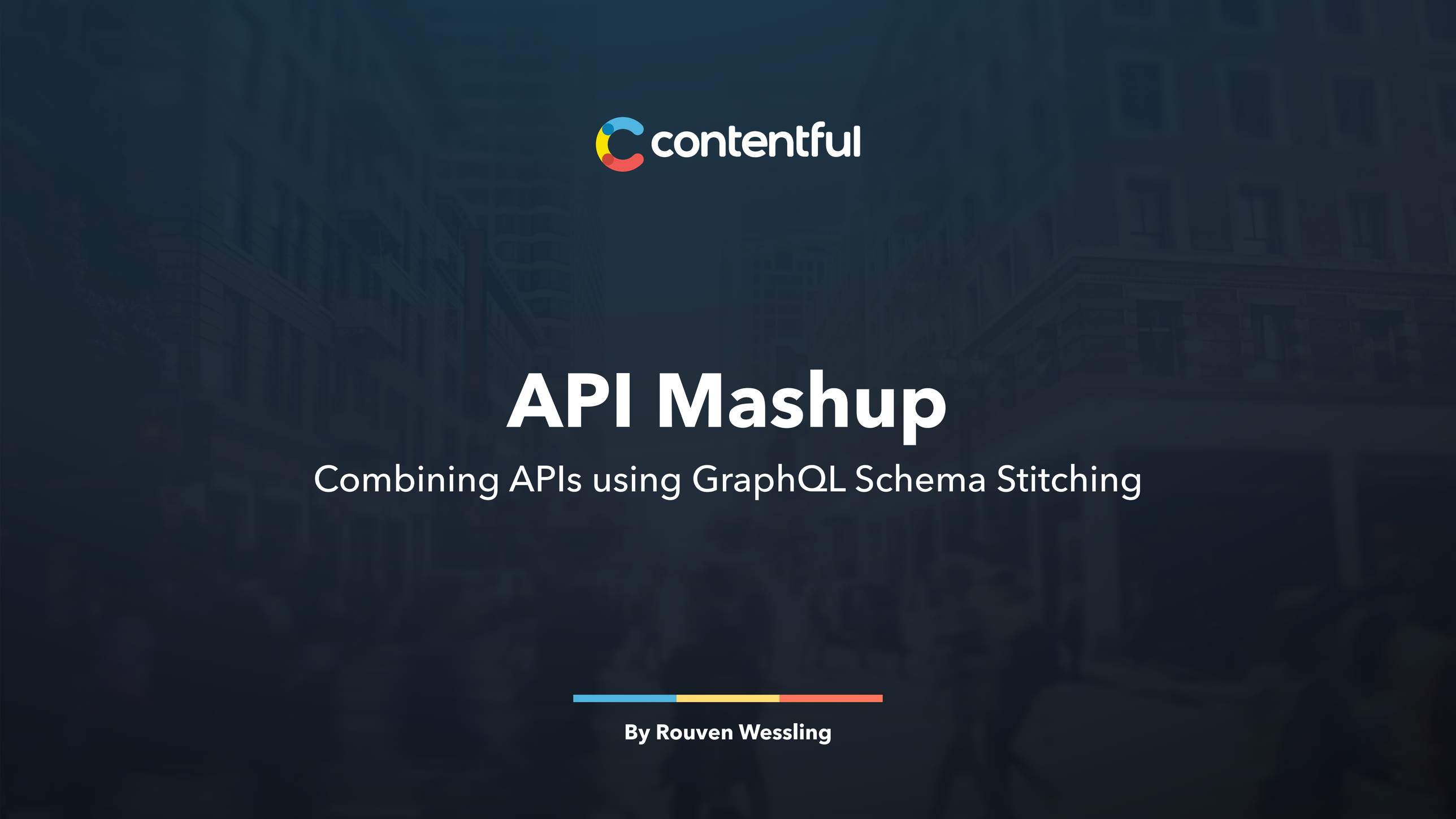


API Mashup

Combining APIs using GraphQL Schema Stitching

By Rouven Wessling

Hi, I'm Rouven

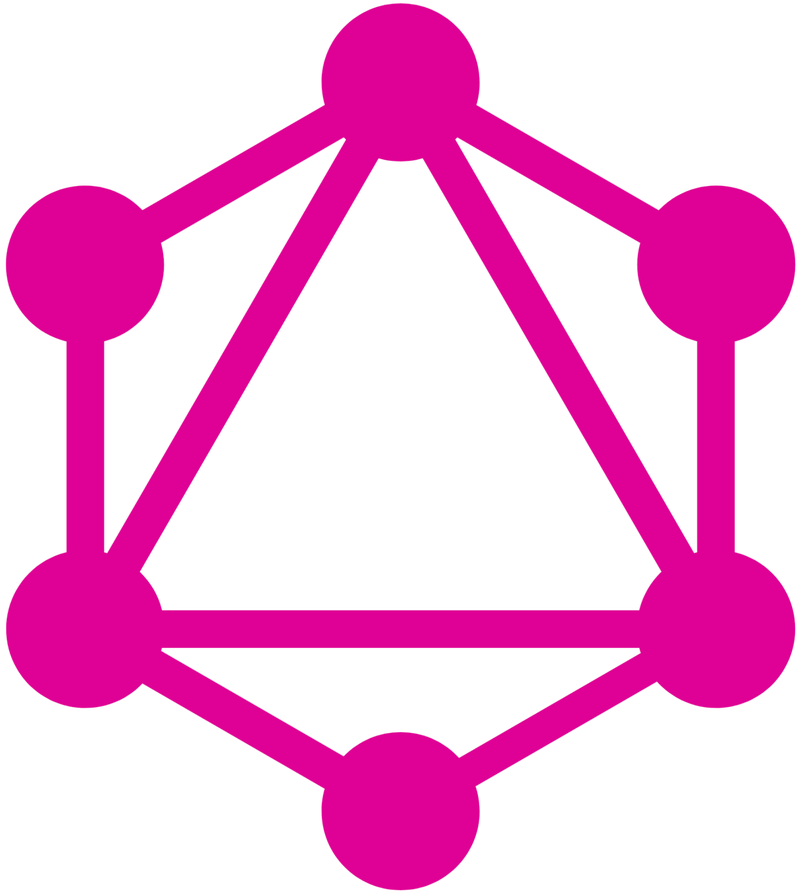
Developer Evangelist, lover of APIs and cake

@RouvenWessling


contentful
Don’t let a CMS get in the way of shipping software.
Contentful provides a content infrastructure that enables teams to power content in any digital product.

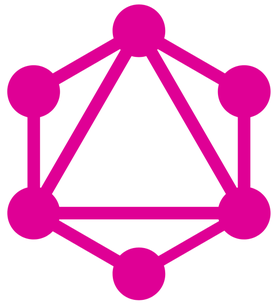

Let’s talk about APIs

Developers ❤️ APIs



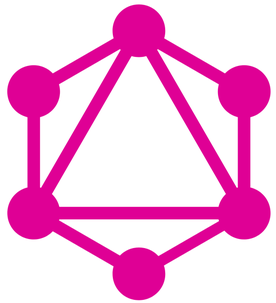
GraphQL

A short introduction



GraphQL Advantages

- Strongly typed schema
- Introspectable
- Get just the data you want
- Get *all* the data you want
- Frontend-driven data selection
- Composable

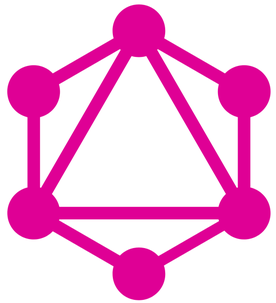


GraphQL

> ***Backend Engineer**: Hmm. So you're saying this "GraphQL" will allow any web or native engineer to arbitrarily query basically any field in any backend service, recursively, however they want, without any backend engineers involved?*

Adam Neary Follow
Software engineer @airbnb. Dad, pizzaiolo.
May 29 · 9 min read

https://medium.com/airbnb-engineering/reconciling-graphql-and-thrift-at-airbnb-a97e8d290712

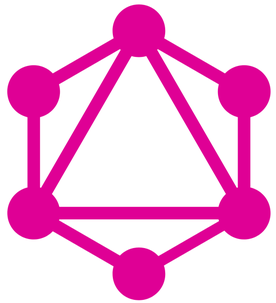


GraphQL

> ***Backend Engineer**: Hmm. So you're saying this "GraphQL" will allow any web or native engineer to arbitrarily query basically any field in any backend service, recursively, however they want, without any backend engineers involved?*
>
> ***Frontend Engineer**: Yeah, right? It's amazing!*

Adam Neary Follow
Software engineer @airbnb. Dad, pizzaiolo.
May 29 · 9 min read

https://medium.com/airbnb-engineering/reconciling-graphql-and-thrift-at-airbnb-a97e8d290712

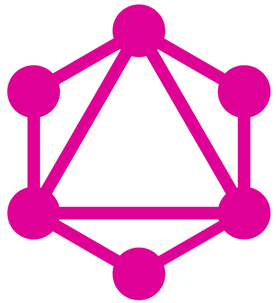


GraphQL

> ***Backend Engineer**: Hmm. So you're saying this "GraphQL" will allow any web or native engineer to arbitrarily query basically any field in any backend service, recursively, however they want, without any backend engineers involved?*
>
> ***Frontend Engineer**: Yeah, right? It's amazing!*
>
> *[…silence…]*

Adam Neary Follow
Software engineer @airbnb. Dad, pizzaiolo.
May 29 · 9 min read

https://medium.com/airbnb-engineering/reconciling-graphql-and-thrift-at-airbnb-a97e8d290712

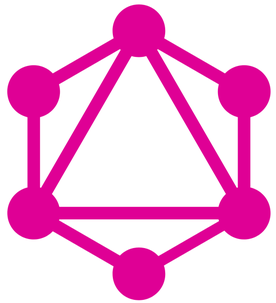


GraphQL

> ***Backend Engineer**: Hmm. So you're saying this "GraphQL" will allow any web or native engineer to arbitrarily query basically any field in any backend service, recursively, however they want, without any backend engineers involved?*
>
> ***Frontend Engineer**: Yeah, right? It's amazing!*
>
> *[…silence…]*
>
> ***Backend Engineer**: Guards, seize this person.*

Adam Neary Follow
Software engineer @airbnb. Dad, pizzaiolo.
May 29 · 9 min read

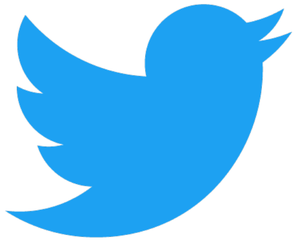
https://medium.com/airbnb-engineering/reconciling-graphql-and-thrift-at-airbnb-a97e8d290712



Who uses GraphQL?

Amplitude

Prisma



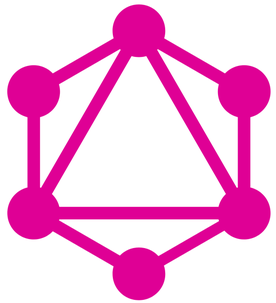
ATLASSIAN

Product Hunt

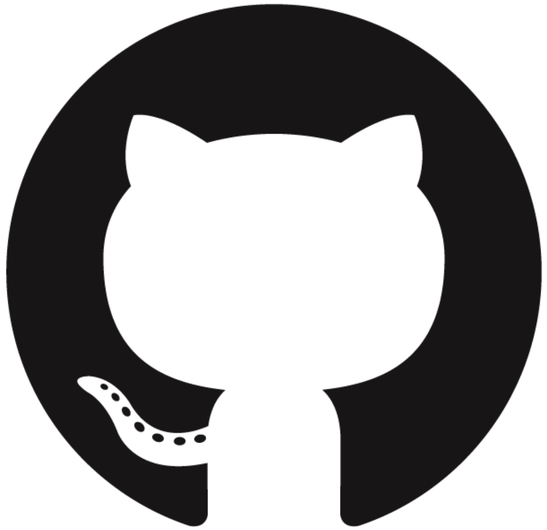
coursera

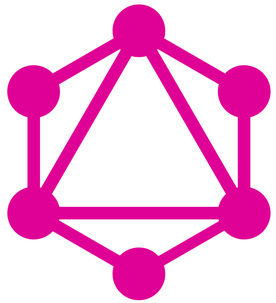
stackshare

KLM



Who uses GraphQL?

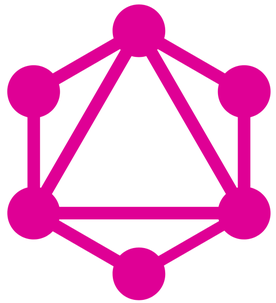




GraphQL Example

```
type Repository implements Node {
    owner: RepositoryOwner!

    url: URI!

    stargazers(
        first: Int

        after: String

        last: Int

        before: String

        orderBy: StarOrder
    ): StargazerConnection!
}
```

Schema



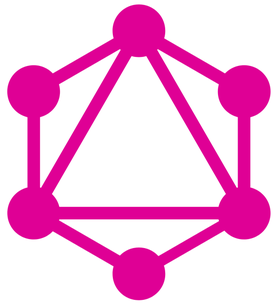
GraphQL Example

```
type Repository implements Node {
    owner: RepositoryOwner!

    url: URI!

    stargazers(
        first: Int

        after: String

        last: Int

        before: String

        orderBy: StarOrder
    ): StargazerConnection!
}
```

Schema

```
query {
    repository(
        owner: "contentful",
        name: "contentful.js"
    ) {
        url
        stargazers {
            totalCount
        }
        owner {
            login
        }
    }
}
```

Query



GraphQL Example

```
type Repository implements Node {
    owner: RepositoryOwner!

    url: URI!

    stargazers(
        first: Int

        after: String

        last: Int

        before: String

        orderBy: StarOrder
    ): StargazerConnection!
}
```

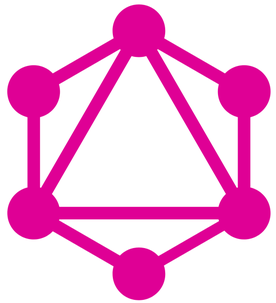
Schema

```
query {
    repository(
        owner: "contentful",
        name: "contentful.js"
    ) {
        url
        stargazers {
            totalCount
        }
        owner {
            login
        }
    }
}
```

Query

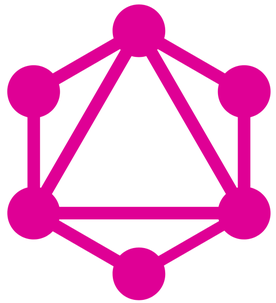
```
{
  "data": {
    "repository": {
      "url": "https://github.com/...",
      "stargazers": {
        "totalCount": 412
      },
      "owner": {
        "login": "contentful"
      }
    }
  }
}
```

Result



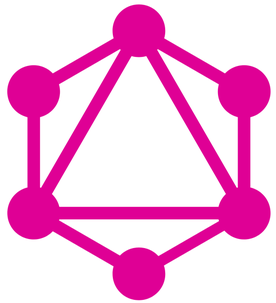
Hello World

```
'use strict';

const {ApolloServer, gql} = require('apollo-server');
```



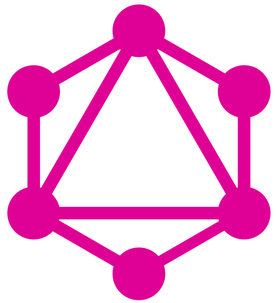
Hello World

```
'use strict';

const {ApolloServer, gql} = require('apollo-server');

// The GraphQL schema
const typeDefs = gql`
    type Query {
        hello: String
    }
`;
```



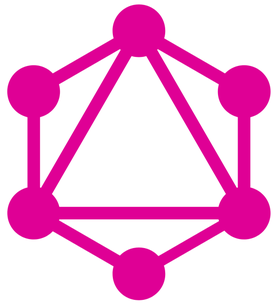
Hello World

```
'use strict';

const {ApolloServer, gql} = require('apollo-server');

// The GraphQL schema
const typeDefs = gql`
    type Query {
        hello: String
    }
`;

// A map of functions which return data for the schema.
const resolvers = {
  Query: {
    hello: () => 'world'
  }
};
```



Hello World

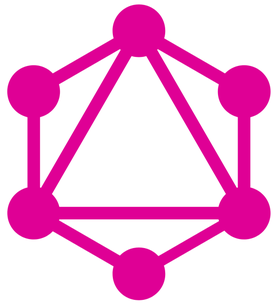
```
'use strict';

const {ApolloServer, gql} = require('apollo-server');

// The GraphQL schema
const typeDefs = gql`
    type Query {
        hello: String
    }
`;

// A map of functions which return data for the schema.
const resolvers = {
  Query: {
    hello: () => 'world'
  }
};

async function startServer() {
  const server = new ApolloServer({ typeDefs, resolvers });

  return await server.listen();
}
```



Hello World

```
type Query {
  hello: String
}
```

Schema



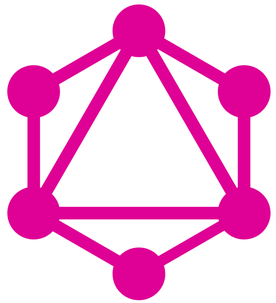
Hello World

```
type Query {
  hello: String
}
```

Schema

```
query {
    hello
}
```

Query



Hello World

```
type Query {
  hello: String
}
```

Schema

```
query {
    hello
}
```

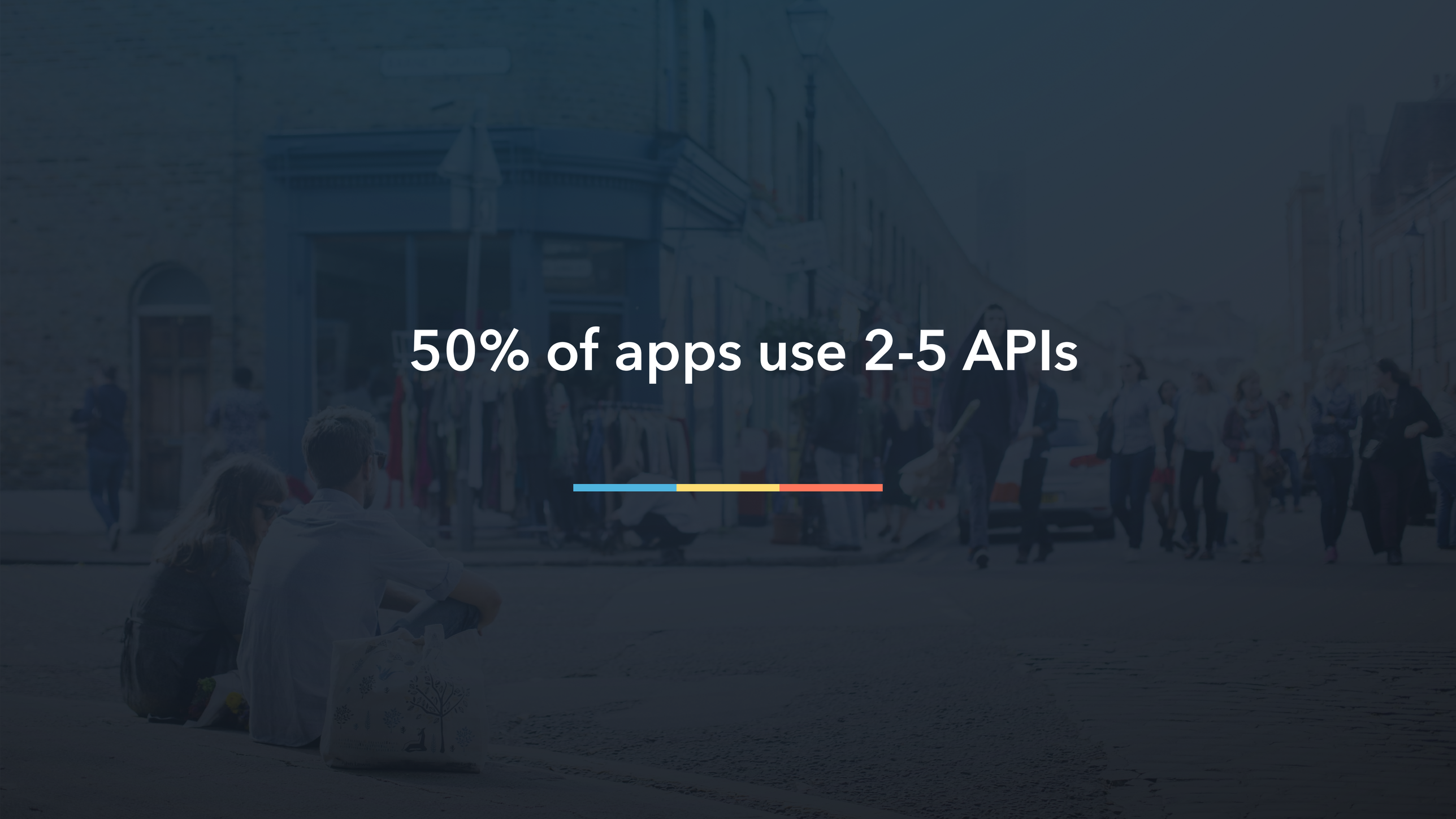
Query

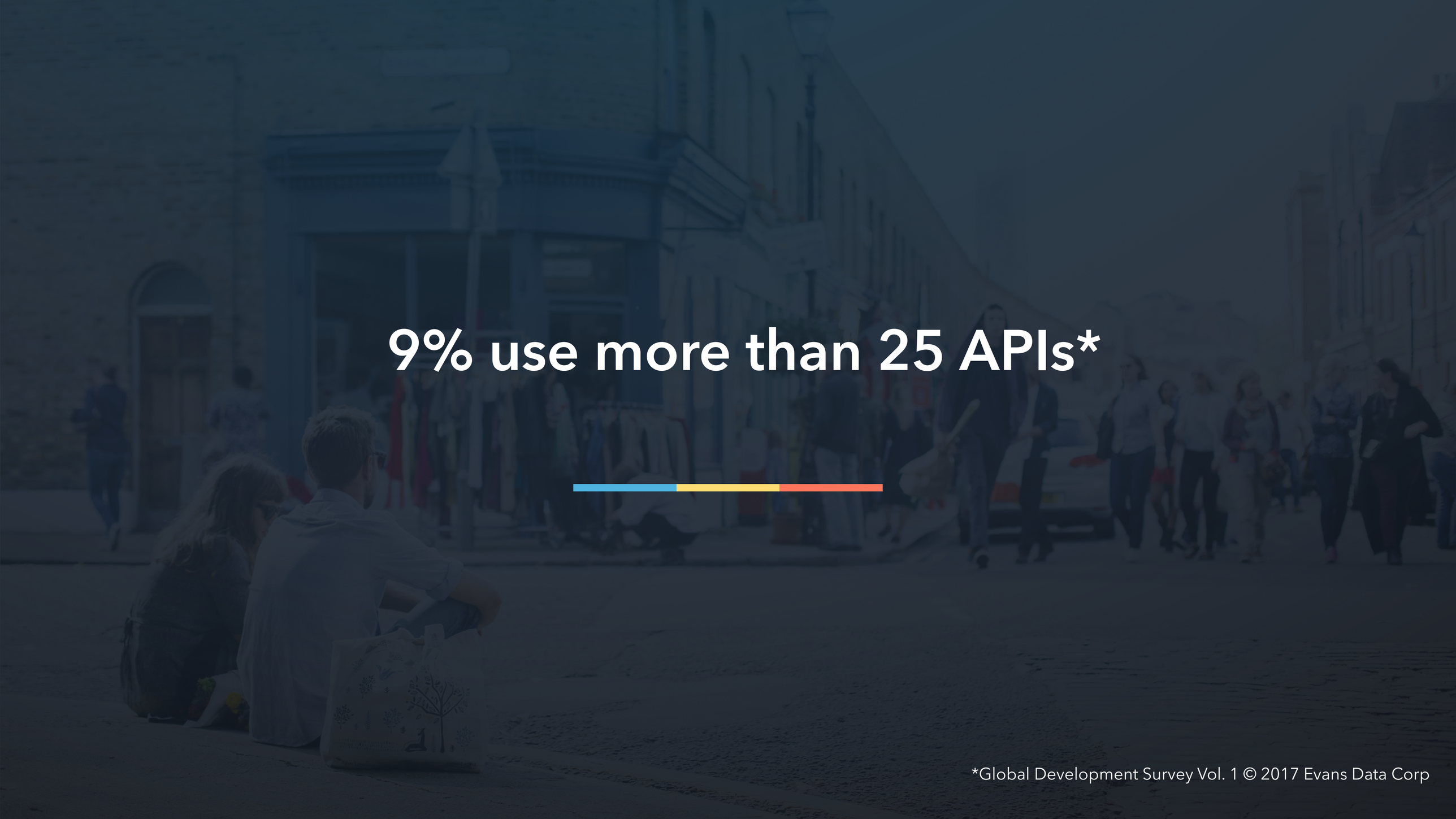
```
{
  "data": {
    "hello": "world"
  }
}
```

Result

Back to the ~~Legos~~ APIs

Developers ❤️ APIs


50% of apps use 2-5 APIs

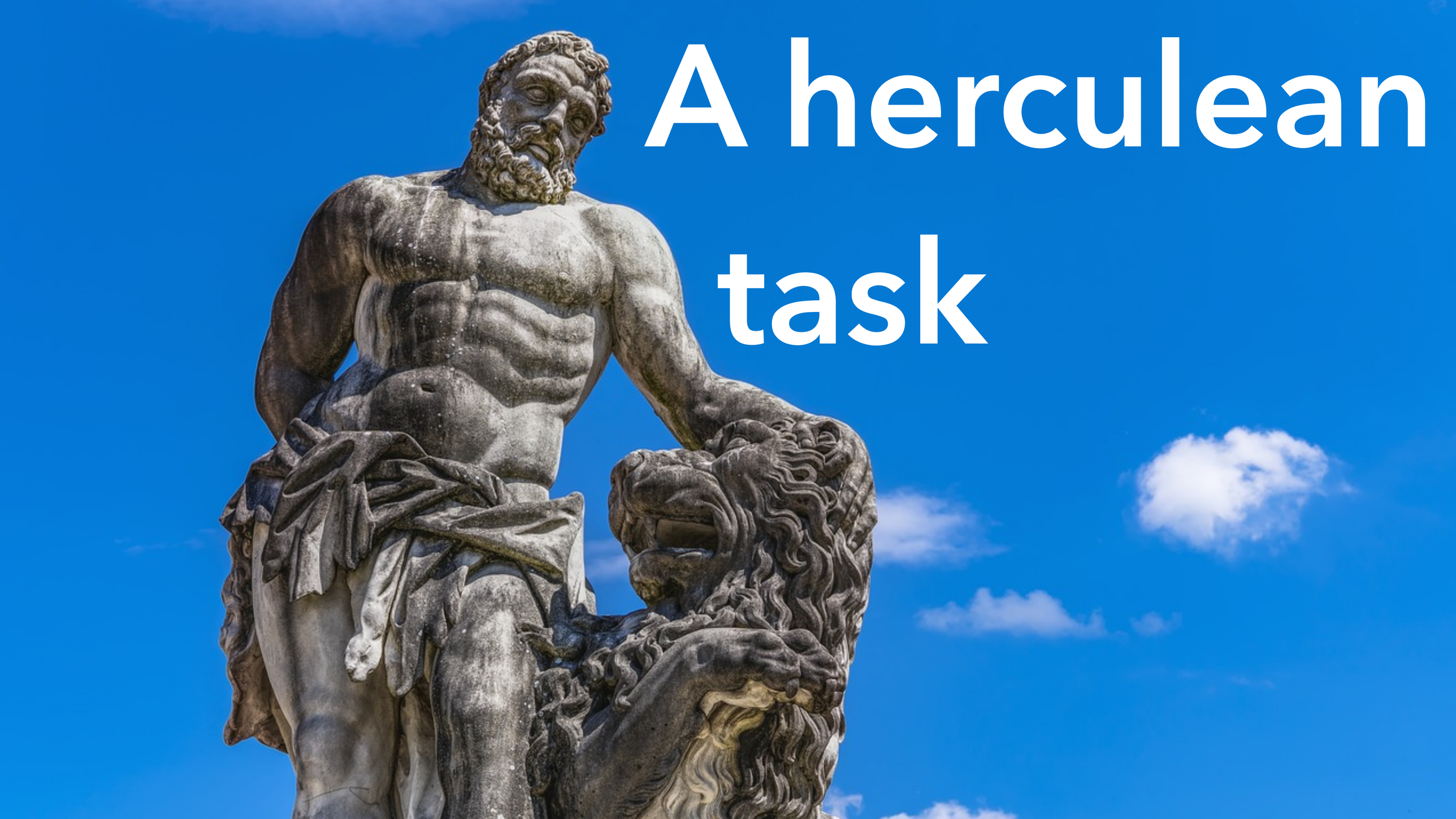


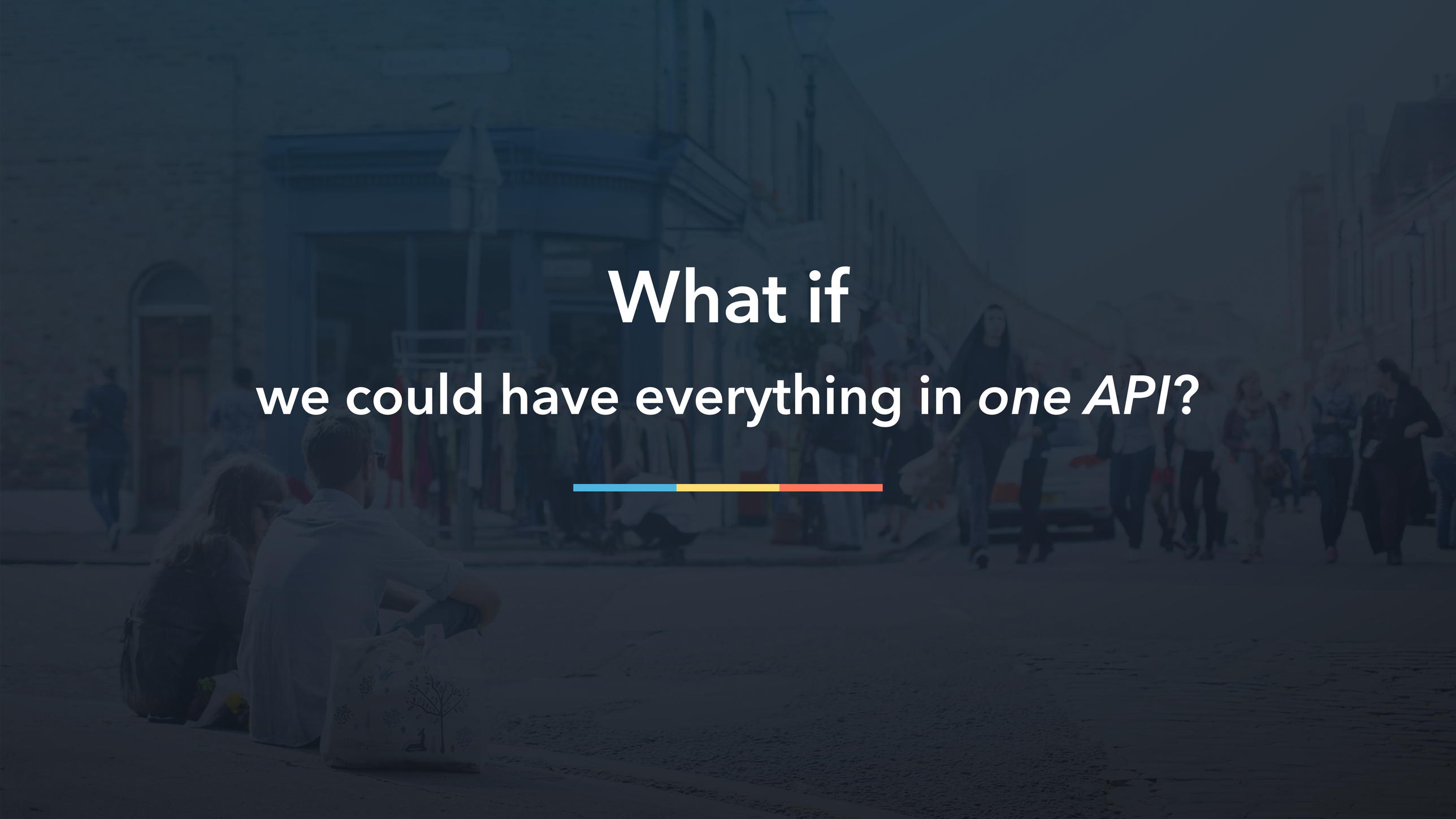
*Global Development Survey Vol. 1 © 2017 Evans Data Corp

Image by Arto Alanenpää CC-BY-SA 4.0 https://commons.wikimedia.org/wiki/File:Lego_dublo_arto_alanenpaa_5.JPG

You need to know…

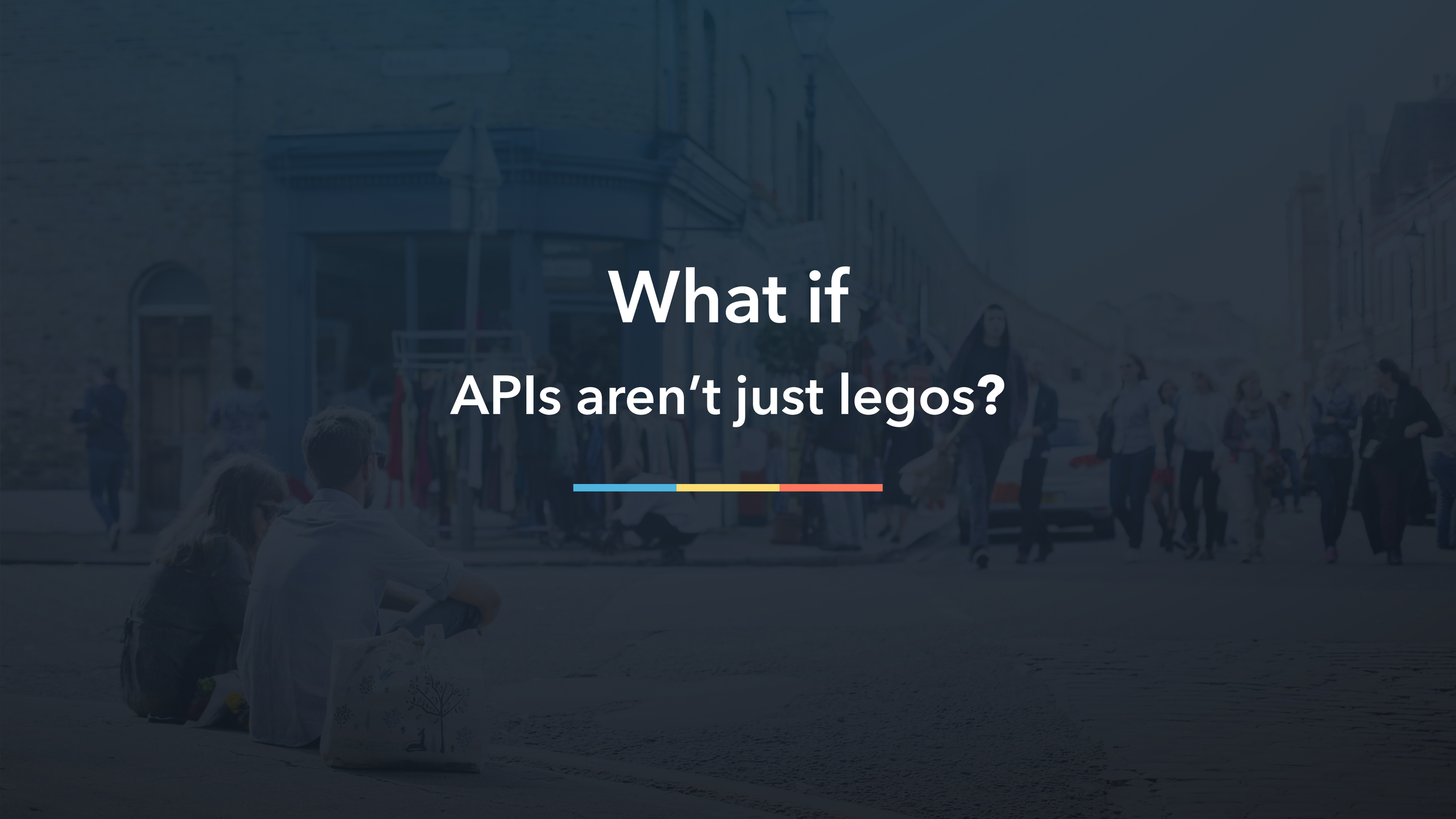
- …which API holds which data
- …how to authenticate against each API
- …which fields indicate a reference to another API


A herculean
task



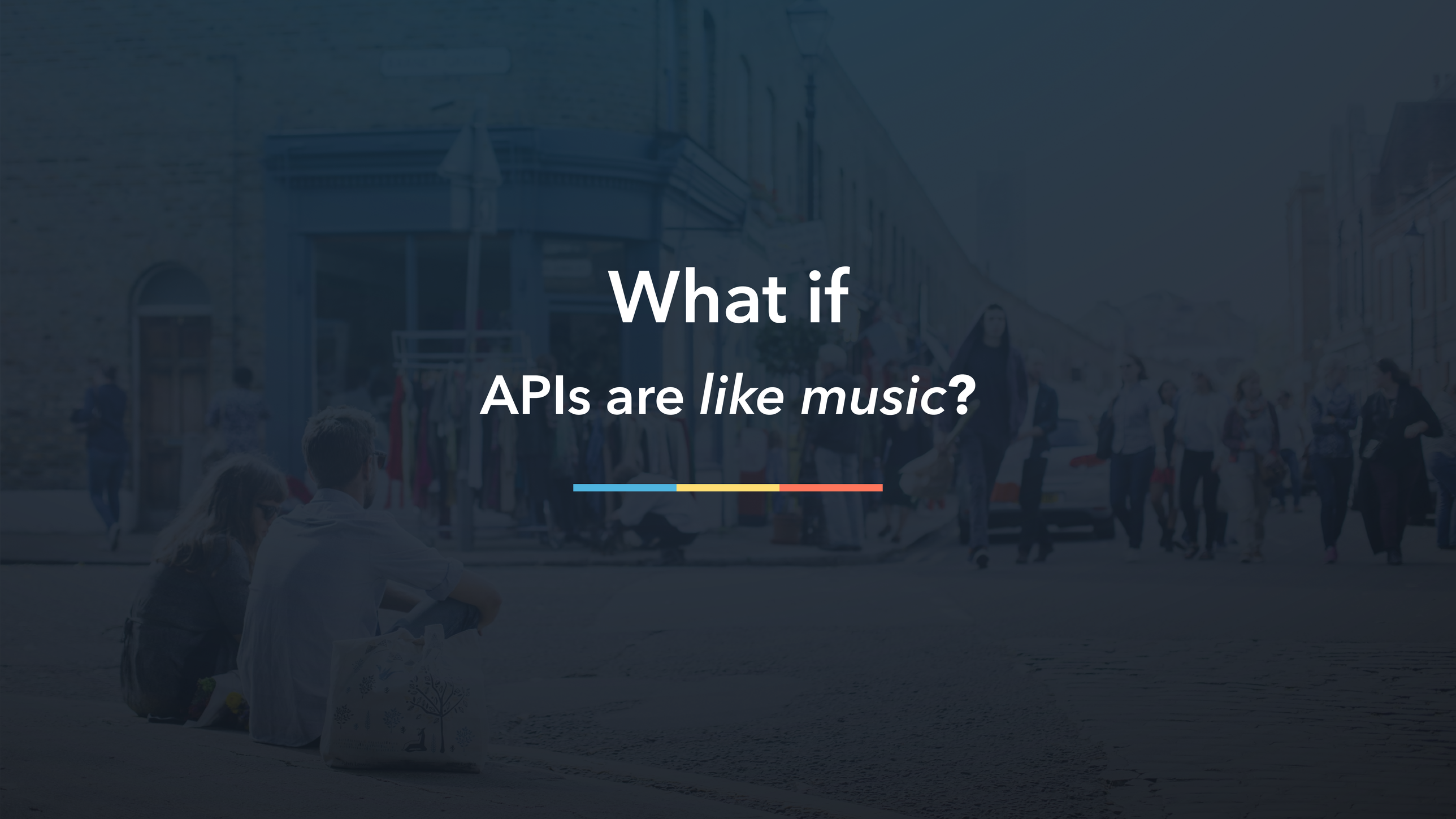
What if

we could have everything in *one API*?



What if

APIs aren’t just legos?



What if

APIs are *like music*?

A mashup

is a song that’s created by blending two existing songs together.

A great mashup

is a good song in itself, but also leaves it sources recognizable .

Say you have two songs

Some indie rock

Say you have two songs

Some indie rock

and an electronic track

Say you have two songs

Some indie rock

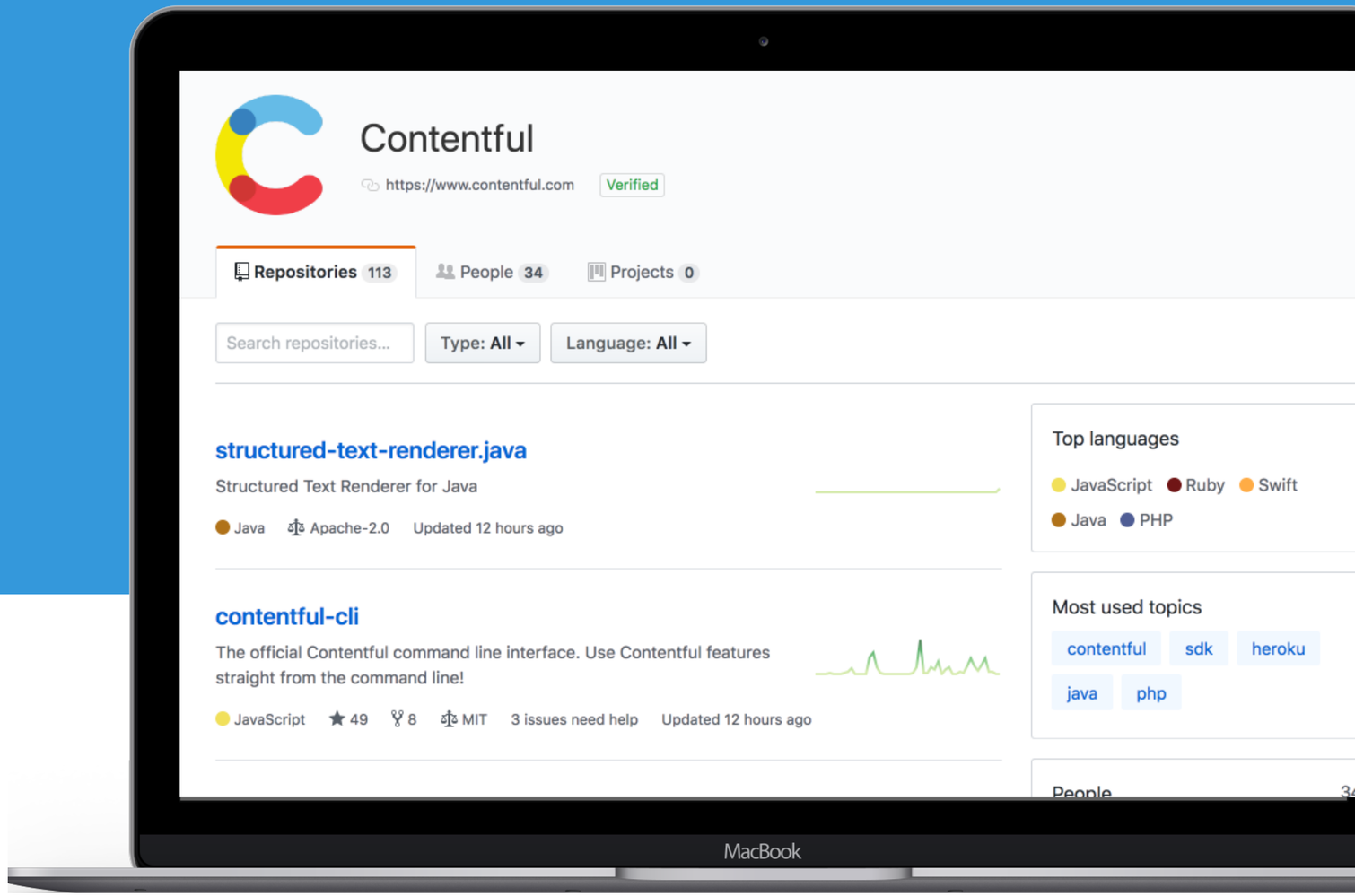
and an electronic track

can make for a great mashup

An Example

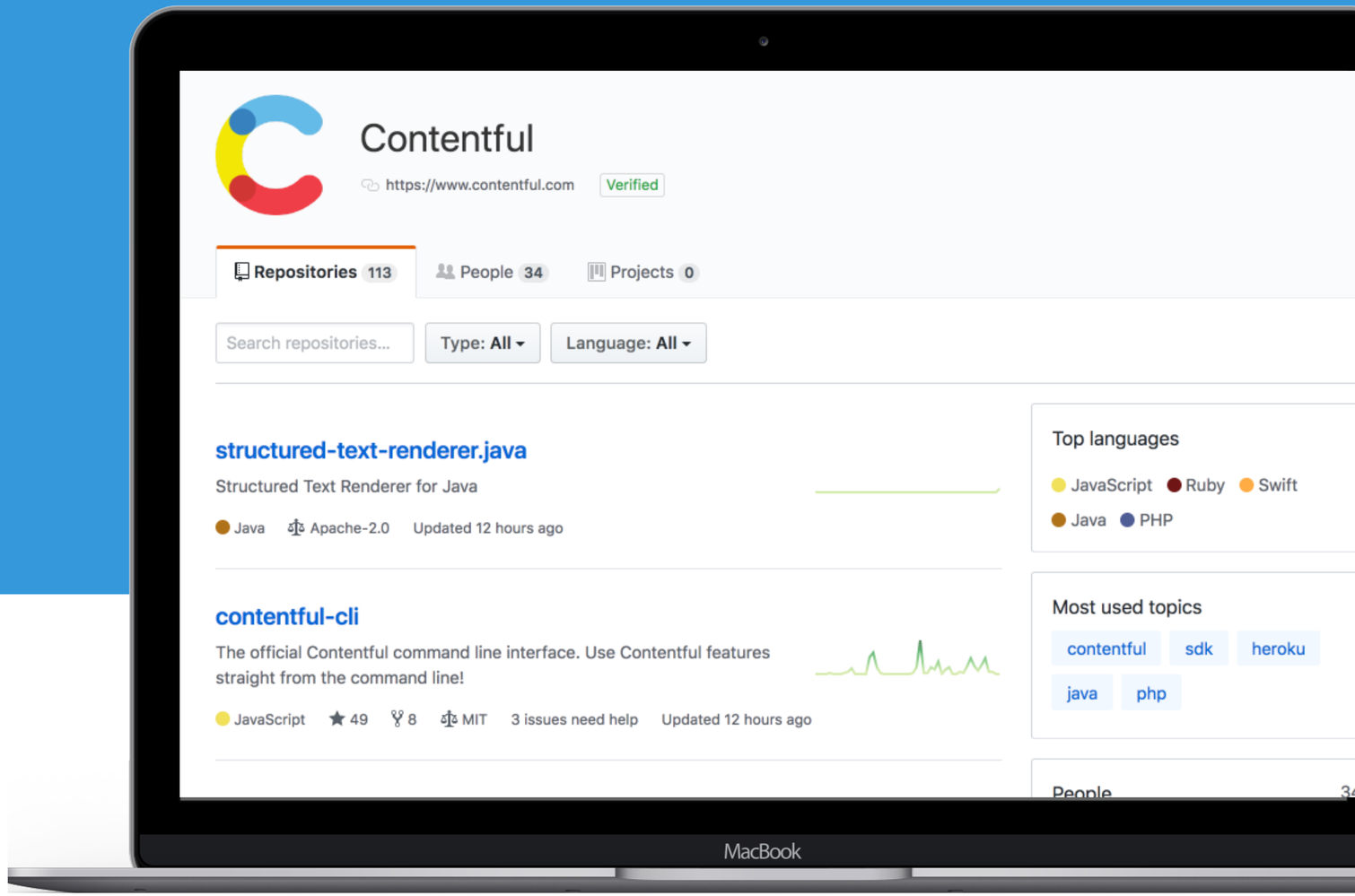
We got a lot of Open Source projects

And it's getting hard to keep track of them.

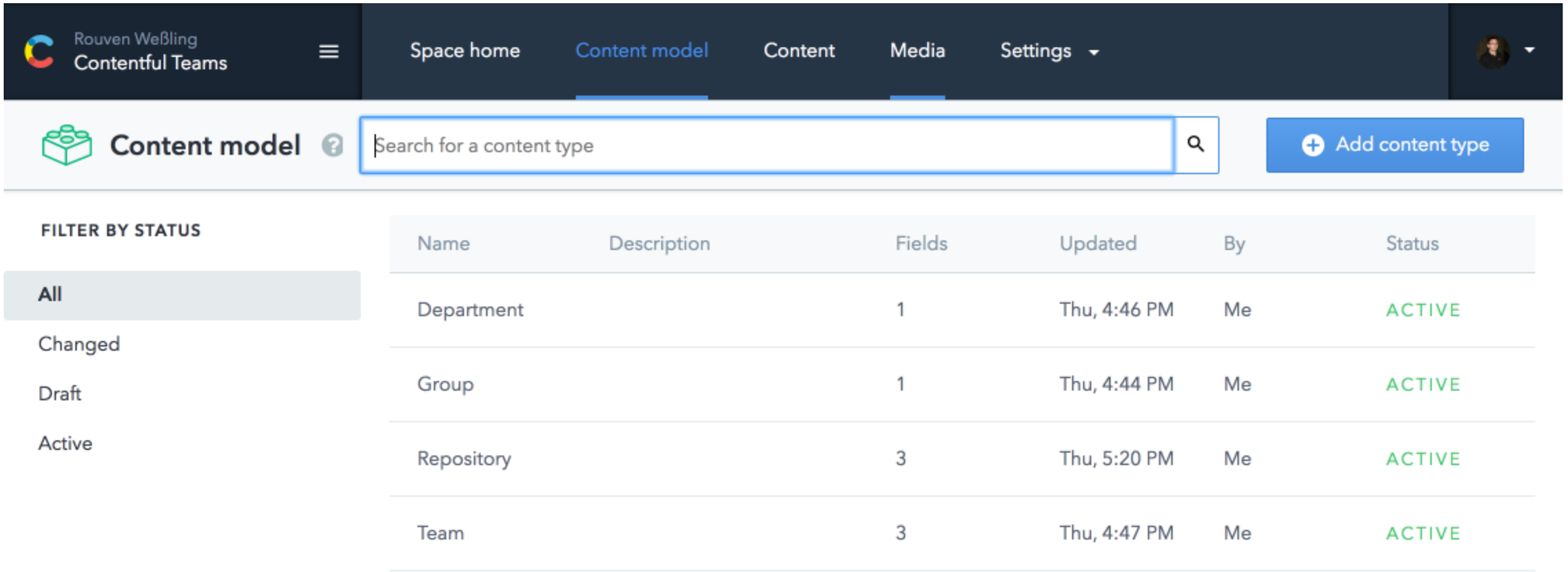


We need a dashboard

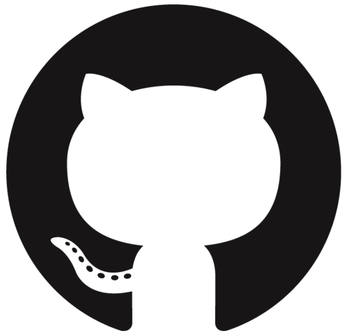
- Pull information from GitHub
- Store metadata in Contentful
- Pull information from other services, e.g. TravisCI



Content model

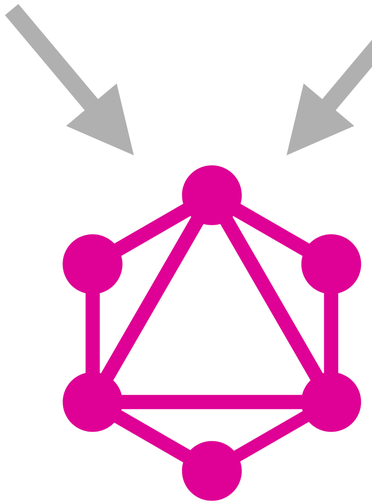


Setup



GitHub

Contentful



Stitching Proxy

Photo by Lindsay Henwood on Unsplash

Step 1

A remote schema

Photo by freestocks.org on Unsplash

A remote schema

```
const {GITHUB_TOKEN} = require('./config.json');

const gitHubLink = setContext((request) => ({
  headers: {
    'Authorization': `Bearer ${GITHUB_TOKEN}`,
  }
})).concat(new HttpLink({uri: 'https://api.github.com/graphql', fetch}));
```

A remote schema

```
const {GITHUB_TOKEN} = require('./config.json');

const gitHubLink = setContext((request) => ({
  headers: {
    'Authorization': `Bearer ${GITHUB_TOKEN}`,
  }
})).concat(new HttpLink({uri: 'https://api.github.com/graphql', fetch}));

async function startServer() {
  const gitHubRemoteSchema = await introspectSchema(gitHubLink);
```

A remote schema

```
const {GITHUB_TOKEN} = require('./config.json');

const gitHubLink = setContext((request) => ({
  headers: {
    'Authorization': `Bearer ${GITHUB_TOKEN}`,
  }
})).concat(new HttpLink({uri: 'https://api.github.com/graphql', fetch}));

async function startServer() {
  const gitHubRemoteSchema = await introspectSchema(gitHubLink);

  const gitHubSchema = makeRemoteExecutableSchema({
    schema: gitHubRemoteSchema,
    link: gitHubLink,
  });
```

A remote schema

```
const {GITHUB_TOKEN} = require('./config.json');

const gitHubLink = setContext((request) => ({
  headers: {
    'Authorization': `Bearer ${GITHUB_TOKEN}`,
  }
})).concat(new HttpLink({uri: 'https://api.github.com/graphql', fetch}));

async function startServer() {
  const gitHubRemoteSchema = await introspectSchema(gitHubLink);

  const gitHubSchema = makeRemoteExecutableSchema({
    schema: gitHubRemoteSchema,
    link: gitHubLink,
  });

  const server = new ApolloServer({schema: gitHubSchema});

  return await server.listen();
}

startServer().then(({ url }) => {
  console.log(`🚀 Server ready at ${url}`);
});
```

¿Por qué no los dos?

Step 2

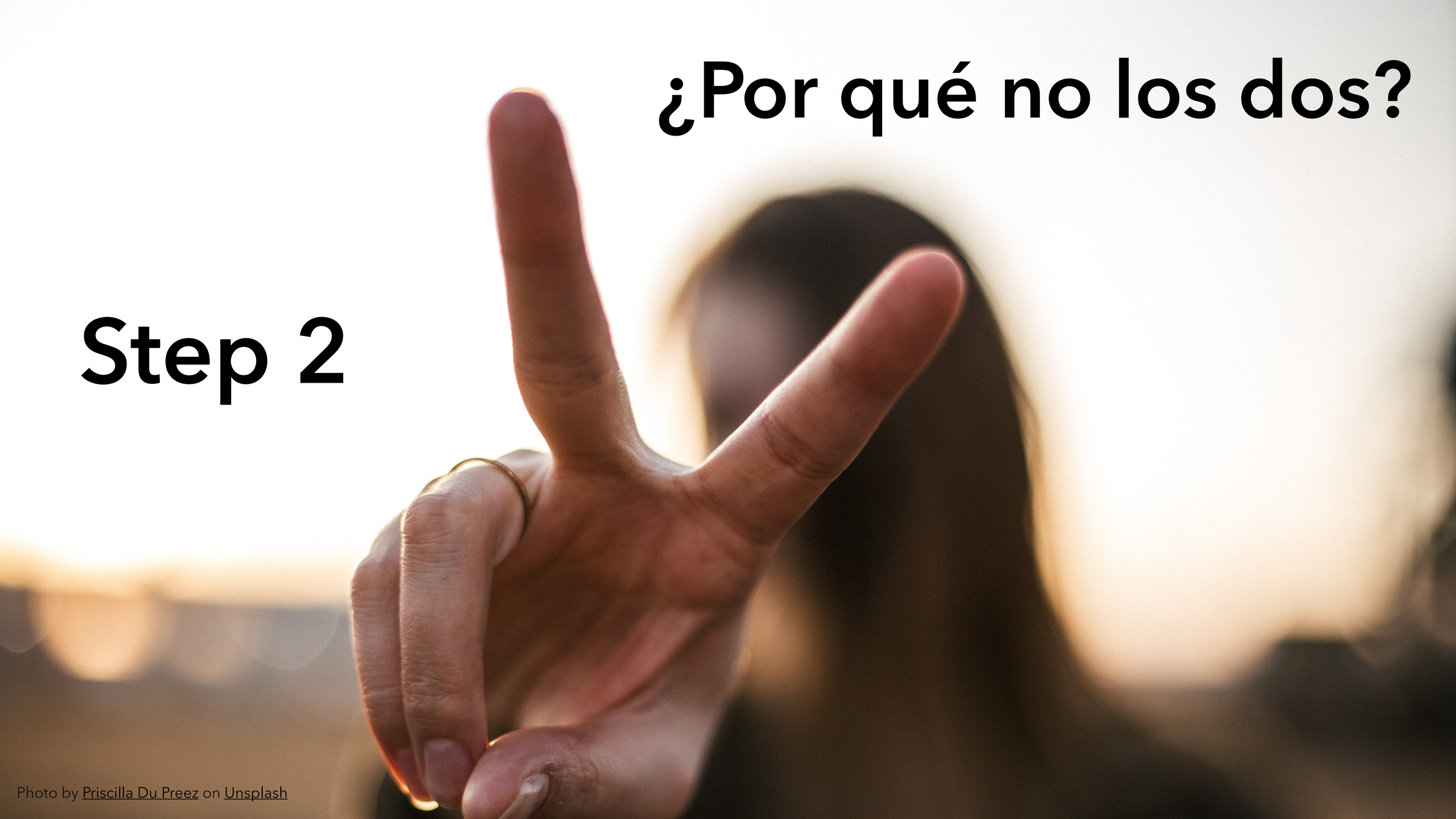


Photo by Priscilla Du Preez on Unsplash

¿Por qué no los dos?

```
const {mergeSchemas} = require('graphql-tools');

const getGitHubSchema = require('./github.js');
const getContentfulSchema = require('./contentful.js');
```

¿Por qué no los dos?

```
const {mergeSchemas} = require('graphql-tools');

const getGitHubSchema = require('./github.js');
const getContentfulSchema = require('./contentful.js');

async function startServer() {
  const gitHubSchema = await getGitHubSchema();
  const contentfulSchema = await getContentfulSchema();
```

¿Por qué no los dos?

```
const {mergeSchemas} = require('graphql-tools');

const getGitHubSchema = require('./github.js');
const getContentfulSchema = require('./contentful.js');

async function startServer() {
  const gitHubSchema = await getGitHubSchema();
  const contentfulSchema = await getContentfulSchema();

  const schema = mergeSchemas({
    schemas: [
      gitHubSchema,
      contentfulSchema
    ]
  });
```

¿Por qué no los dos?

```
const {mergeSchemas} = require('graphql-tools');

const getGitHubSchema = require('./github.js');
const getContentfulSchema = require('./contentful.js');

async function startServer() {
  const gitHubSchema = await getGitHubSchema();
  const contentfulSchema = await getContentfulSchema();

  const schema = mergeSchemas({
    schemas: [
      gitHubSchema,
      contentfulSchema
    ]
  });

  const server = new ApolloServer({schema});

  return await server.listen();
}

startServer().then(({ url }) => {
  console.log(`🚀 Server ready at ${url}`);
});
```

Schemas

```
type Repository implements Node {
    owner: RepositoryOwner!

    url: URI!

    stargazers(
        first: Int

        after: String

        last: Int

        before: String

        orderBy: StarOrder
    ): StargazerConnection!
}
```

GitHub

```
type Repository implements Entry {
    gitHubLogin: String

    gitHubName: String

    owningTeam(
        preview: Boolean,

        locale: String
    ): Team
}
```

Contentful

Problem

Photo by Joao Tzanno on Unsplash

Our schemas conflict

```
type Repository implements Node {
    owner: RepositoryOwner!

    url: URI!

    stargazers(
        first: Int

        after: String

        last: Int

        before: String

        orderBy: StarOrder
    ): StargazerConnection!
}
```

GitHub

```
type Repository implements Entry {
    gitHubLogin: String

    gitHubName: String

    owningTeam(
        preview: Boolean,

        locale: String
    ): Team
}
```

Contentful

Photo by Suzanne D. Williams on Unsplash

Schema transformation

```
const {
  introspectSchema,
  makeRemoteExecutableSchema,
  transformSchema,
  RenameTypes,
  RenameRootFields
} = require('graphql-tools');

async function getContentfulSchema() {
  const contentfulRemoteSchema = await introspectSchema(contentfulLink);

  const contentfulSchema = makeRemoteExecutableSchema({
    schema: contentfulRemoteSchema,
    link: contentfulLink,
  });

  return transformSchema(contentfulSchema, [
```

Schema transformation

```
const {
  introspectSchema,
  makeRemoteExecutableSchema,
  transformSchema,
  RenameTypes,
  RenameRootFields
} = require('graphql-tools');

async function getContentfulSchema() {
  const contentfulRemoteSchema = await introspectSchema(contentfulLink);

  const contentfulSchema = makeRemoteExecutableSchema({
    schema: contentfulRemoteSchema,
    link: contentfulLink,
  });

  return transformSchema(contentfulSchema, [
    new RenameTypes((name) => {
      if (name.includes('Repository')) {
        return name.replace('Repository', 'RepositoryMetadata');
      }

      return name;
    }),
```

Schema transformation

```
const {
  introspectSchema,
  makeRemoteExecutableSchema,
  transformSchema,
  RenameTypes,
  RenameRootFields
} = require('graphql-tools');

async function getContentfulSchema() {
  const contentfulRemoteSchema = await introspectSchema(contentfulLink);

  const contentfulSchema = makeRemoteExecutableSchema({
    schema: contentfulRemoteSchema,
    link: contentfulLink,
  });

  return transformSchema(contentfulSchema, [
    new RenameTypes((name) => {
      if (name.includes('Repository')) {
        return name.replace('Repository', 'RepositoryMetadata');
      }

      return name;
    }),
    new RenameRootFields((operation, fieldName) => {
      if (fieldName.includes('repository')) {
        return fieldName.replace('repository', 'repositoryMetadata');
      }

      return fieldName;
    })
  ]);
}
```

Schemas transformed

```
type Repository implements Node {
    owner: RepositoryOwner!

    url: URI!

    stargazers(
        first: Int

        after: String

        last: Int

        before: String

        orderBy: StarOrder
    ): StargazerConnection!
}
```

GitHub

```
type RepositoryMetadata implements Entry {
    gitHubLogin: String

    gitHubName: String

    owningTeam(
        preview: Boolean,

        locale: String
    ): Team
}
```

Contentful


Step 4
Snitches get stitches

Snitches get stitches

```
const linkTypeDefs = `
extend type Repository {
  contentfulMetadata: RepositoryMetadata
}

extend type RepositoryMetadata {
  gitHubRepository: Repository
}
`;
```

Snitches get stitches

```
RepositoryMetadata: {
  gitHubRepository: {
```

Snitches get stitches

```
RepositoryMetadata: {
  gitHubRepository: {
    fragment: '... on RepositoryMetadata { gitHubName gitHubLogin }',
    resolve(repositoryMetadata, args, context, info) {
      return info.mergeInfo.delegateToSchema({
        schema: gitHubSchema,
```

Snitches get stitches

```
RepositoryMetadata: {
  gitHubRepository: {
    fragment: '... on RepositoryMetadata { gitHubName gitHubLogin }',
    resolve(repositoryMetadata, args, context, info) {
      return info.mergeInfo.delegateToSchema({
        schema: gitHubSchema,
        operation: 'query',
        fieldName: 'repository',
        args: {
          owner: repositoryMetadata.gitHubLogin,
          name: repositoryMetadata.gitHubName
        },
        context,
        info
      });
    }
  }
}
```

Too Easy?

Photo by Alain d'Alché on Unsplash

Snitches get stitches - part 2

```
Repository: {
  contentfulMetadata: {
```

Snitches get stitches - part 2

```
Repository: {
  contentfulMetadata: {
    fragment: '... on Repository { name owner { login } }',
```

Snitches get stitches - part 2

```
Repository: {
  contentfulMetadata: {
    fragment: '... on Repository { name owner { login } }',
```

Snitches get stitches - part 2

```
Repository: {
  contentfulMetadata: {
    fragment: '... on Repository { name owner { login } }',
    resolve(repository, args, context, info) {
      return info.mergeInfo.delegateToSchema({
        schema: transformedContentfulSchema,
        operation: 'query',
        fieldName: 'repositoryMetadataCollection',
        args: {
          where: {
            gitHubLogin: repository.owner.login,
            gitHubName: repository.name
          },
          limit: 1
        },
        context,
        info,
```

Snitches get stitches - part 2

```
        transforms: [
          new WrapQuery(
            ['repositoryMetadataCollection'],
            (subtree) => ({
                kind: Kind.FIELD,
                name: {
                  kind: Kind.NAME,
                  value: 'items'
                },
                selectionSet: subtree,
            }),
            result => result && result.items && result.items[0]
          )
        ]
      });
    }
  }
},
```

Schemas stitched

```
type Repositoty implements Node {
    owner: RepositoryOwner!

    url: URI!

    contentfulMetadata: RepositoryMetadata

    stargazers(
        first: Int

        after: String

        last: Int

        before: String

        orderBy: StarOrder
    ): StargazerConnection!
}
```

GitHub

```
type RepositoryMetadata implements Entry {
    gitHubLogin: String

    gitHubName: String

    gitHubRepository: Repository

    owningTeam(
        preview: Boolean,

        locale: String
    ): Team
}
```

Contentful

Bonus

Enrich data

Photo by Boudewijn Huysmans on Unsplash

Enrich data

```
const expandGitHubTypeDefs = `
extend type Repository {
  hasTravisCi: Boolean!
}
`;
```

Enrich data

```
Repository: {
  hasTravisCi: {
    fragment: '... on Repository { travisYml: object(expression:"master:.travis.yml") {... on Blob { text } } }',
    resolve(repository) {
      return Boolean(repository.travisYml && repository.travisYml.text.length > 0);
    }
  }
},
```

Demo

Stop treating your APIs like silos.

Photo by David Pennington on Unsplash


Use schema stitching to allow developers to seamlessly access data across APIs.


And make some music.

ROUVEN WEßLING

Twitter: @RouvenWessling

Email: rouven@contentful.com

Come work with me

West Coast Evangelist
based in the SF Bay Area

Email: rouven@contentful.com

ROUVEN WEßLING

Twitter: @RouvenWessling

Email: rouven@contentful.com