

# Optimizing Data Access

For Frontend Use Cases  
With A Modular Api Gateway





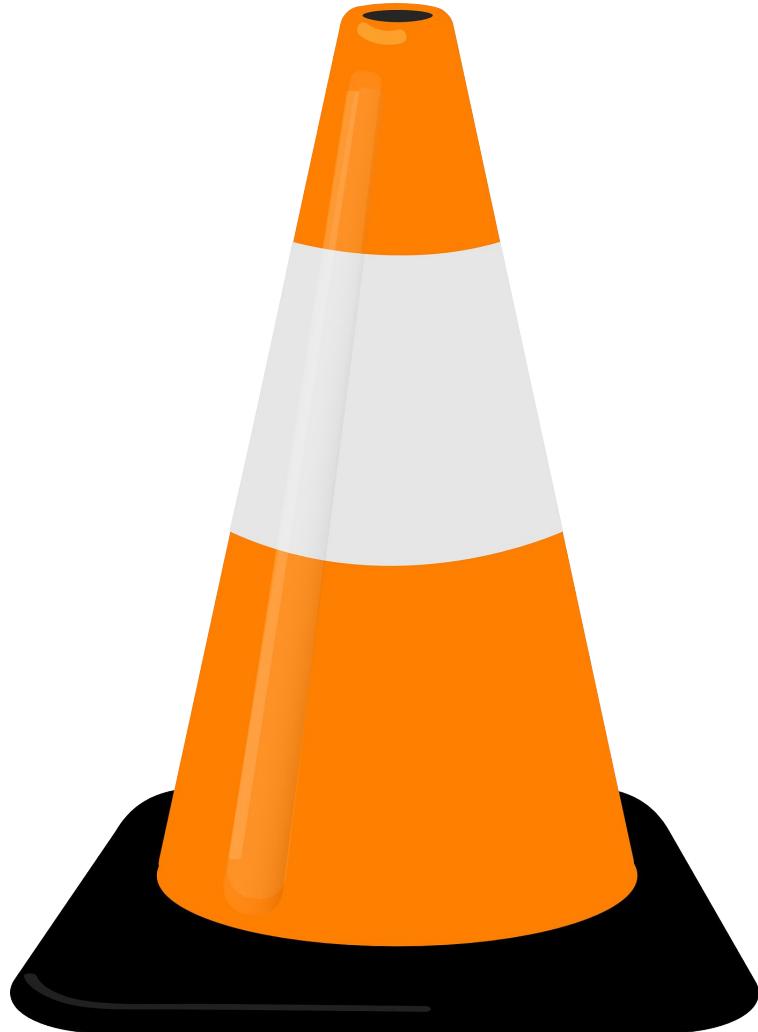
Naming things is hard!

# What I meant was...

Discuss how we can :

Enable frontend developers to access the data they need to *drive their user's experiences*.

The Pattern I'll describe is best known as “Backend for Frontend”.



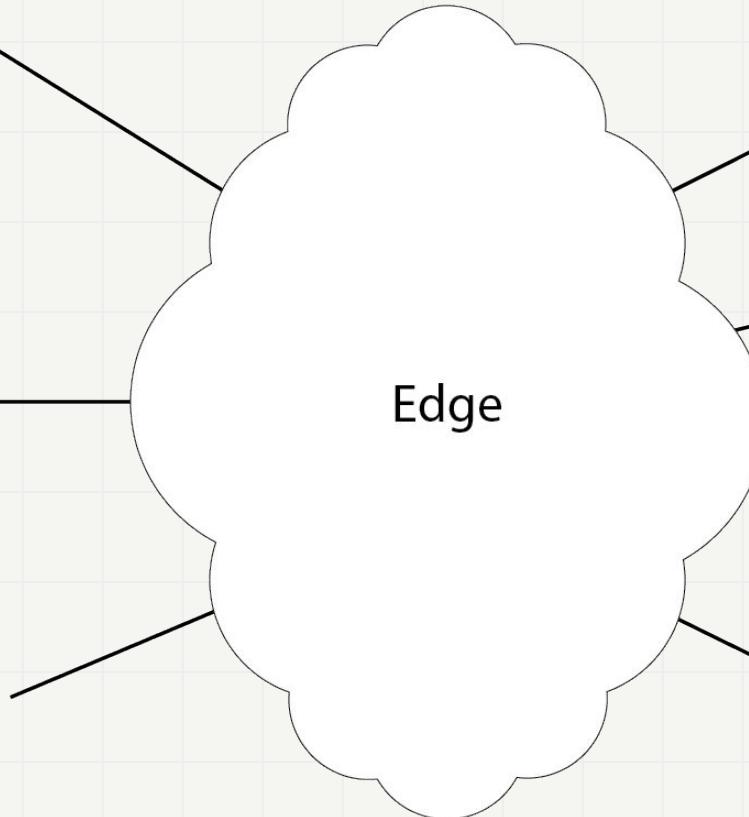
# Who am I?

Brian Leathem

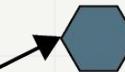
Senior Software Engineer @ Netflix  
- Edge Developer Productivity

@brianleathem

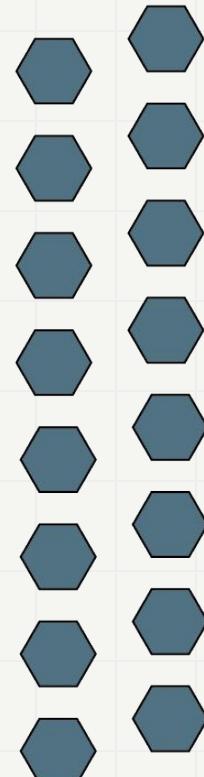




Midtier



Microservices





# The Ideal Developer Experience

Brian Leathem - Optimizing Data Access  
All things Open 2018

New  
Epis  
New  
Epis  
New  
Epis  
New  
Epis  
New  
Epis





# **GreenField: A developer's bliss**

No legacy codebase

No technical debt

No features to support

Other than those ahead of you!

Rapid turnaround!

- Run the whole stack on your laptop
- Easy debugging



# N



# The good old days

And yet our front end developer  
yearns for more simple times...

When she could:

- Work independently
- Achieve a high velocity
- Debug a simple stack
- Run everything locally



# NETFLIX@Scale

# Traffic



# NETFLIX



# Microservices



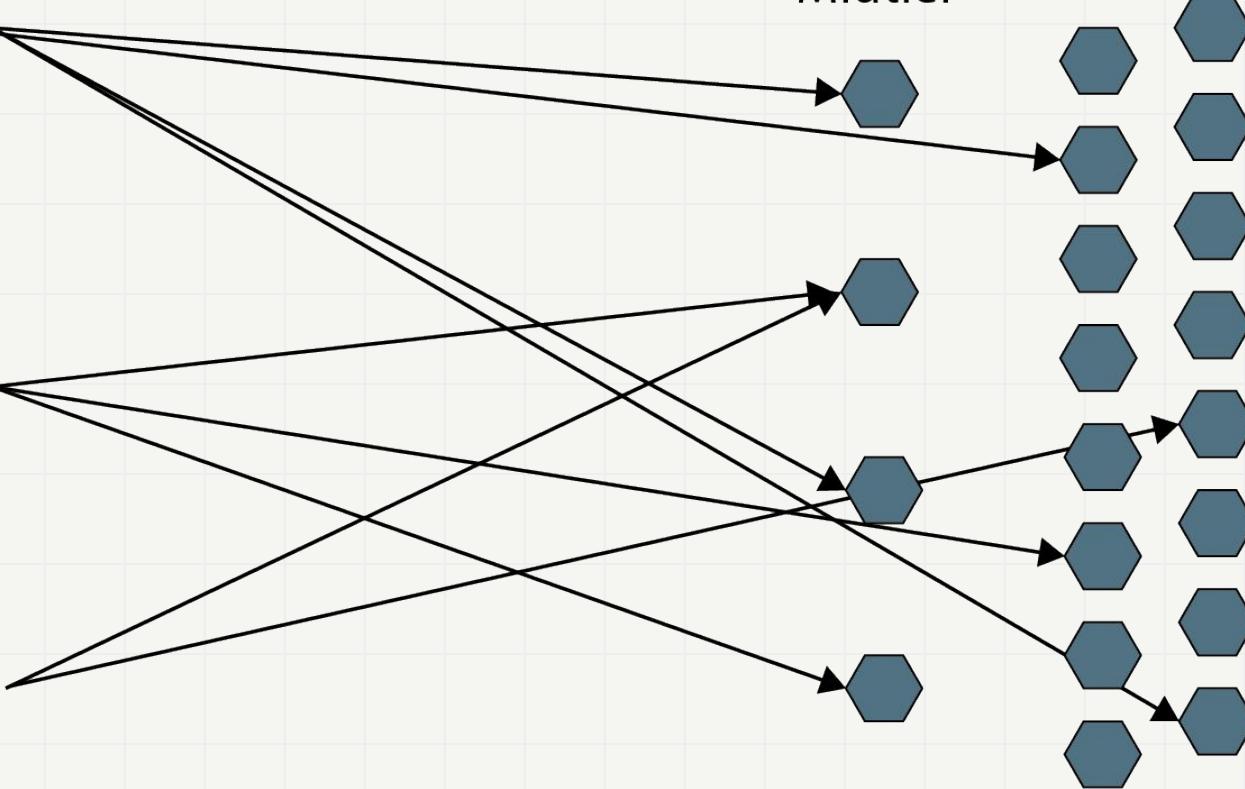


# Team

# Backend For Frontend (BFF)

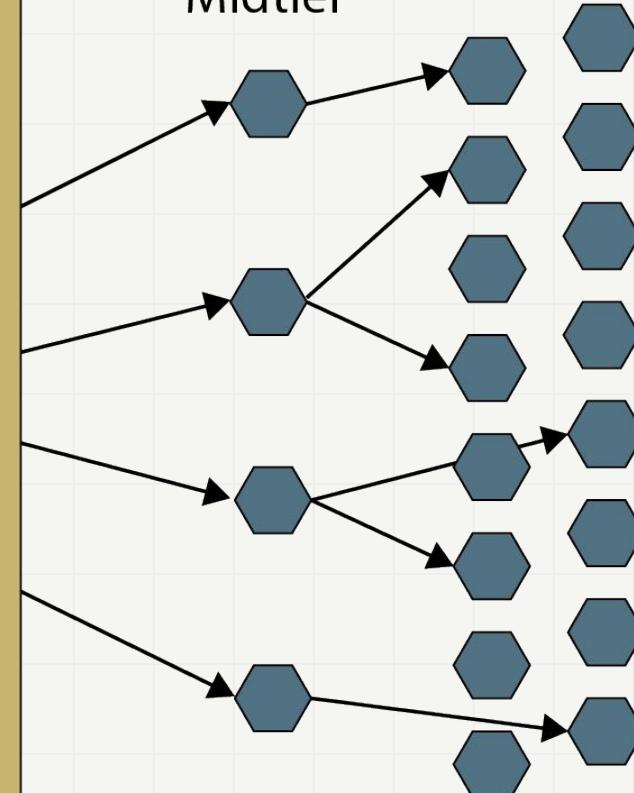
# Microservices

## Midtier



Microservices

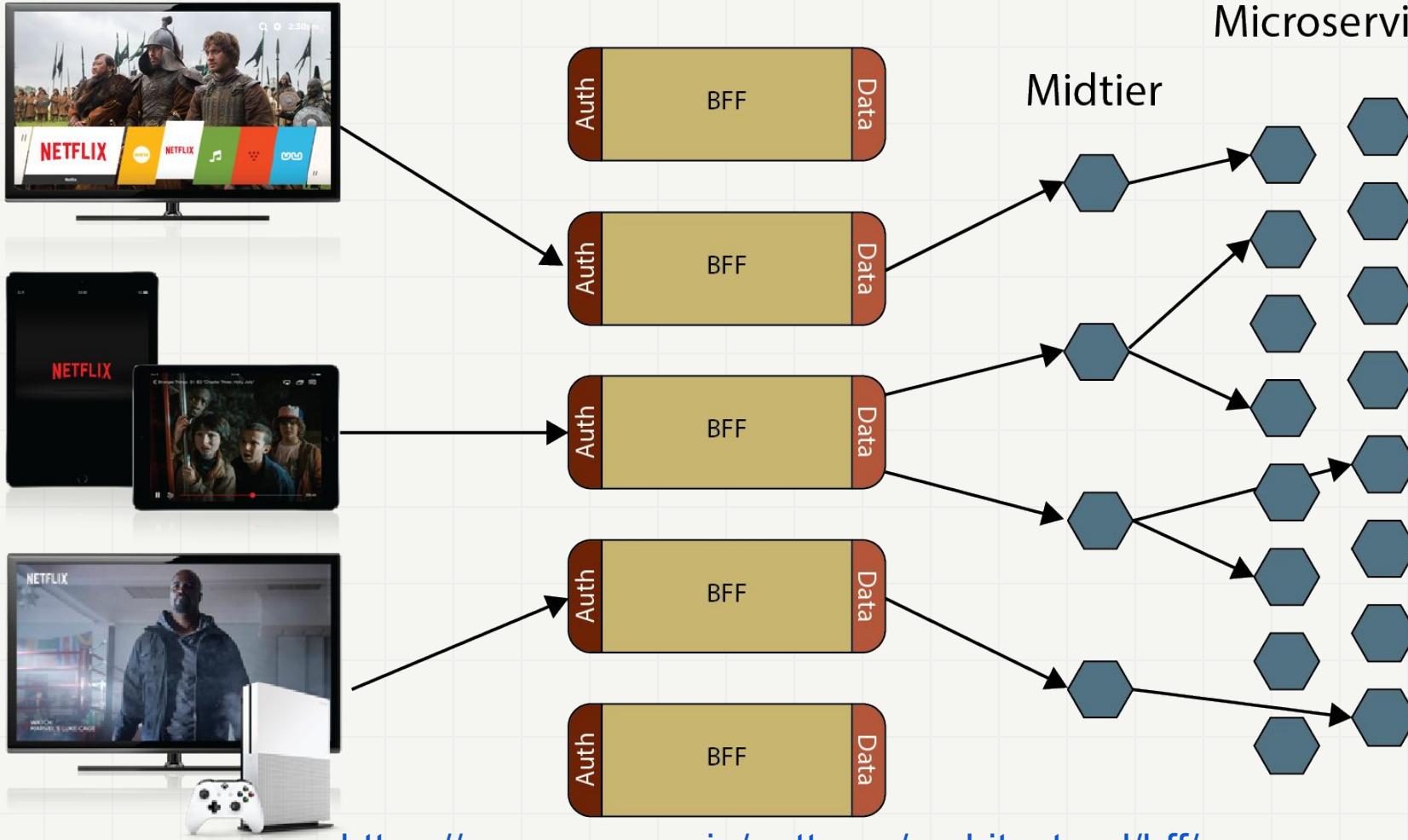
Midtier



General Purpose API Gateway



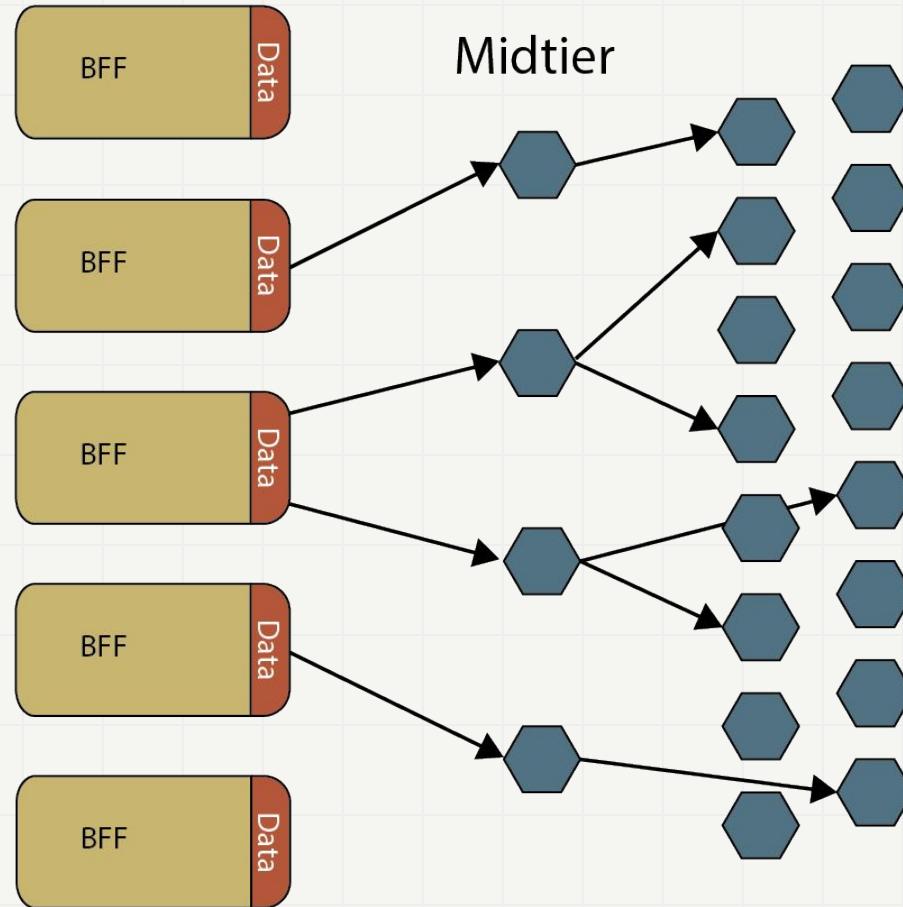
# Microservices



# Microservices



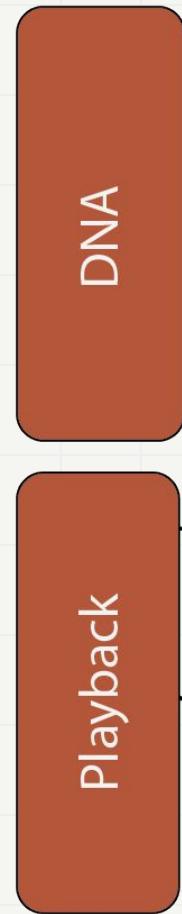
Auth (Zuul)



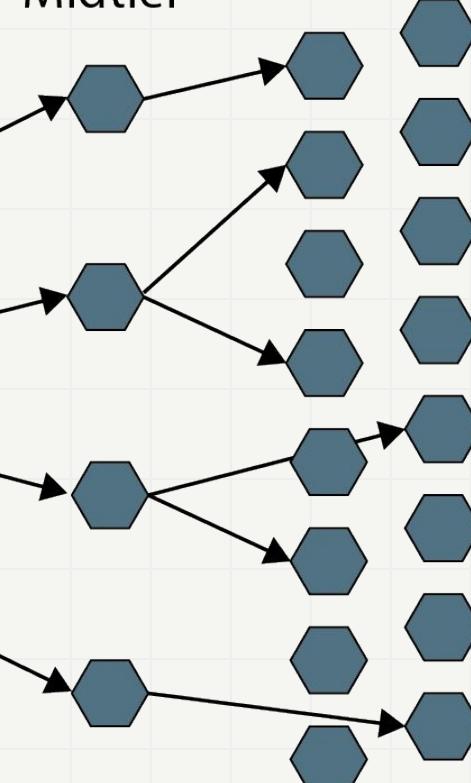
# Microservices



Auth (Zuul)



Midtier



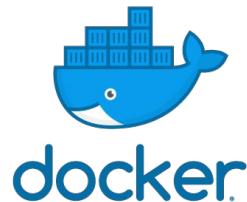
# How we Built our BFF



docker

# Inside the container

- [Node.js](#)
- [Restify](#) for HTTP
- [Eureka](#) for service registration and discovery
- [Archaius](#) for configuration management
- [Atlas](#) for metrics



Application Endpoints

NodeQuark  
- Restify

Node.js 10.x

Linux Base Image (Xenial)

# The Client Data Request

Requirements:

- Fine grained data access
- Aggregate multiple paths into a single network call
- Fault tolerant responses

REST isn't a good fit.

Falcor: distributed JSON graph

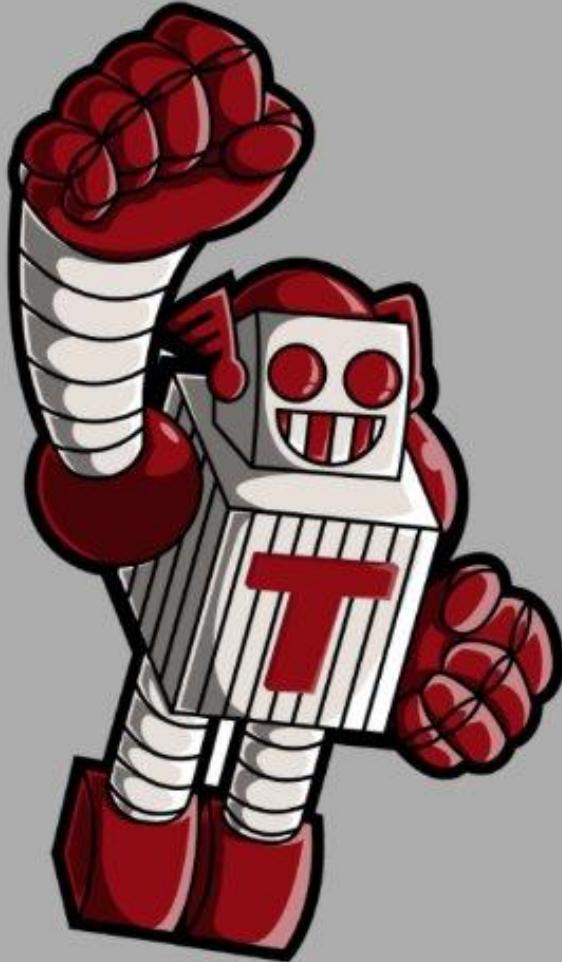
<https://netflix.github.io/falcor/>



# Scaling traffic with Titus

- A production ready container platform
- Integration with AWS
- Netflix OSS integration

<https://netflix.github.io/titus/>

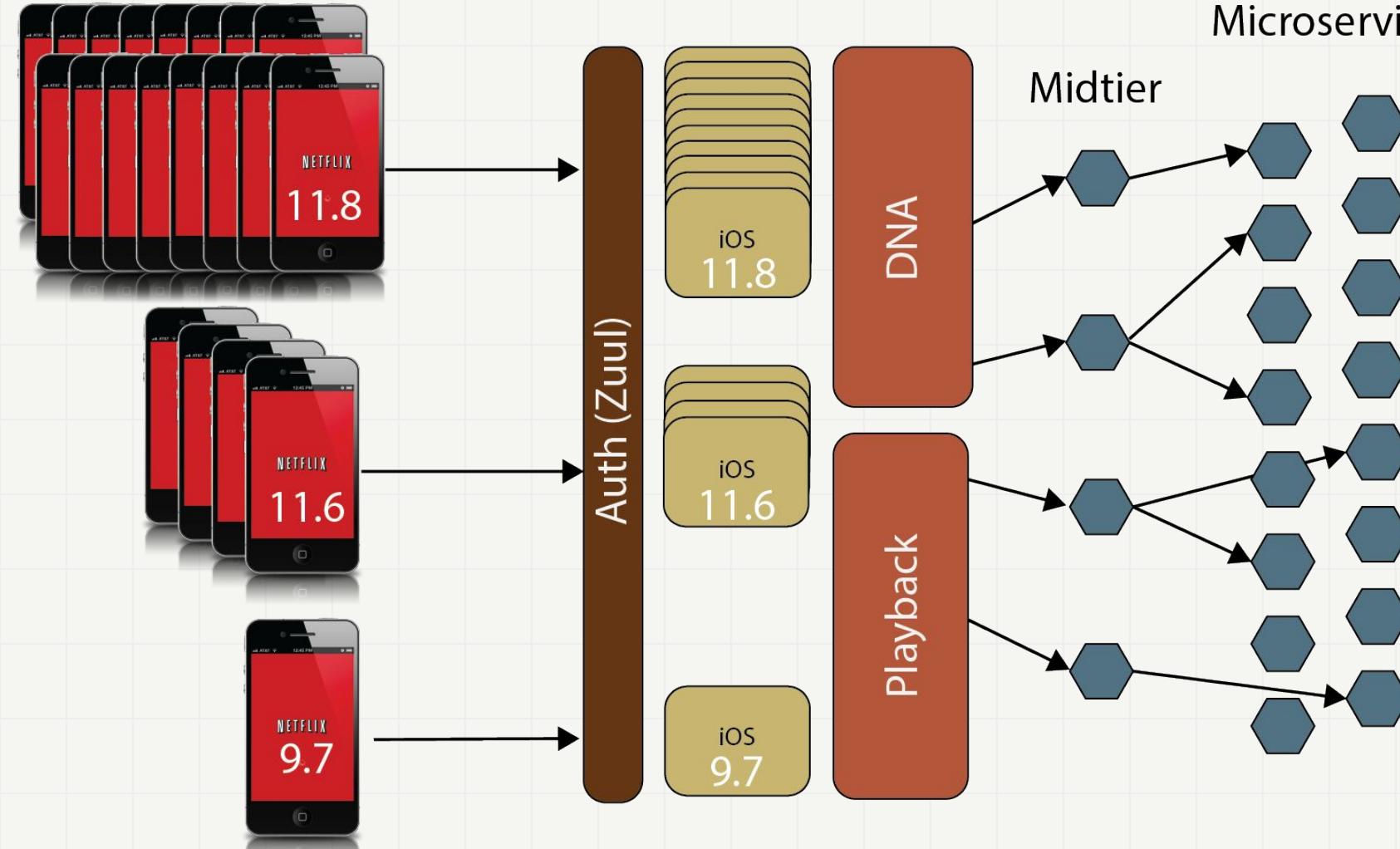


# Dynamic Routing

Dynamic Routing with Zuul  
<https://github.com/Netflix/zuul>



# Microservices



**argo**

IOSUI



DEPLOYMENTS	PUBLICATIONS	CANARY EXECUTIONS	RECENT ACTIVITY	TASKS	CONFIG
STACK	PROD 54	STAGING 84	INT 36	TEST 35	

PROD	Version Constraint	Region	Instance Counts	Created	Image Version	Dash
<b>~11.6.0</b>						
↳	us-east-1	0▲ 0▼ 0△ 0△	2018-10-17 10:27:11 PDT	11.6.58		
↳	us-west-2	0▲ 0▼ 0△ 0△	2018-10-17 12:00:57 PDT	11.6.58		
↳	eu-west-1	0▲ 0▼ 0△ 0△	2018-10-17 15:00:50 PDT	11.6.58		
<b>~11.8.0</b>						
↳	us-east-1	0▲ 0▼ 0△ 0△	2018-10-15 10:35:44 PDT	11.8.32		
↳	us-west-2	0▲ 0▼ 0△ 0△	2018-10-15 10:35:44 PDT	11.8.32		
↳	eu-west-1	0▲ 0▼ 0△ 0△	2018-10-15 10:35:44 PDT	11.8.32		
<b>~11.7.0</b>						
↳	us-east-1	0▲ 0▼ 0△ 0△	2018-10-11 10:21:59 PDT	11.7.44		
↳	us-west-2	0▲ 0▼ 0△ 0△	2018-10-11 10:21:59 PDT	11.7.44		
↳	eu-west-1	0▲ 0▼ 0△ 0△	2018-10-11 10:22:00 PDT	11.7.44		
<b>~11.5.0</b>						

# Scaling Versions

Multiple deployed versions of a BFF endpoint to support

Client has a hardcoded endpoint

Deploy patch releases to a given endpoint



The diagram consists of several text elements arranged vertically. On the far left, the text "Auth (Zuul)" is written vertically in white. To its right, the letters "A/B" are displayed in large, bold black font. Below "A/B", the word "Blue" is written in blue, and below that, the word "Green" is written in green. Above "Blue", there is a plus sign ("+"). A yellow arrow points from the top towards the "A/B" text.

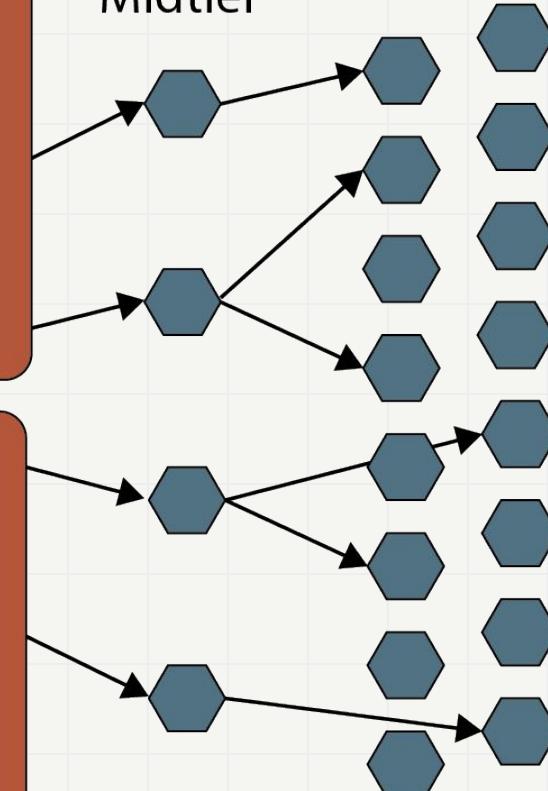


DNA

Playback

## Midtier

# Microservices





# The Developer Experience

# The Developers Goal

Map client requests into backend requests

From the client:

- Path requests from a Falcor client

From the backend:

- Use the DNA API to retrieve the required data



# Sample Client Code

```
function getShowInfo(context) {
  const paths = [
    ['videos', 'showId', 'title'],
    ['currentProfile', 'preferredExperience'],
    ['videos', 'showId', 'seasons', 'length'],
    ['videos', 'showId', 'seasons', {to:9}, ['number', 'numberLabel', 'title', 'id']],
    ['videos', 'showId', 'seasons', {to:9}, 'episodes', 'length'],
    ['videos', 'showId', 'seasons', {to:9}, 'episodes', {to:30},
      ['summary', 'volatile', 'downloadAssetDetails']]
  ]
};

return model.get(paths);
}
```

# Server-side Route Implementation

```
{  
  pattern: ['videos', integerKey('videoId'), 'title'],  
  get({ path, params }, cb) {  
    const query = api  
      .videos(params.videoId)  
      .pluckTitle();  
  
    this.client.get(query, function clientDone(err, response) {  
      if (err) {  
        return cb(PathTerminationError({ cause: err }));  
      }  
      const { data } = response.get(query);  
      return cb(null, { path: value: _.get(data, 'title') });  
    });  
  }  
},
```

# Data Discovery: Comprehensive API docs

Map client requests into backend  
requests

From the client:

- Path requests from a Falcor client

From the backend:

- Use the DNA.js API to retrieve the required API

[Back to EDX-dnajs-release-prepare](#)

[index](#)

[Home](#)  
[Project Documentation](#)  
[REPL](#)  
[DNA Falcor Schema](#)

## CLASSES

[Ab](#)  
[AbTest](#)  
[AbTestList](#)  
[AbTests](#)  
[Account](#)  
[AnnotatedItem](#)  
[AnnotatedItemList](#)  
[AnnotatedList](#)  
[Api](#)  
[Bookmark](#)  
[CategoriesList](#)  
[Category](#)  
[CategoryList](#)  
[Character](#)  
[CharacterArtwork](#)  
[CharacterGallery](#)  
[Episode](#)  
[Geo](#)  
[ImageAtom](#)  
[ImageList](#)  
[InterestingMomentAtom](#)

**pluckRestUri()** → {[Video](#)}

Source: [Video.js](#), line 357

Pluck the `restUri` attribute.

Returns:

Type [Video](#)

**pluckShortSynopsis()** → {[Video](#)}

Source: [Video.js](#), line 365

Pluck the `shortSynopsis` attribute.

Returns:

Type [Video](#)

**pluckShortTitle()** → {[Video](#)}

Source: [Video.js](#), line 373

Pluck the `shortTitle` attribute.

Returns:

Type [Video](#)

**pluckSynopsis()** → {[Video](#)}

Source: [Video.js](#), line 381

# Data Discovery: Online REPL

- Typesafe autocomplete
- Execution against live data
  - Instant feedback
  - Error reporting

## REPL input:

Select a dna.js code snippet, or create your own query expression:

Select a code snippet... ▾

```
1 // Current Profile
2 const query = api.profileCurrent()
3 .pluckName()
4 .pluckMaturityLevel()
5 .
6   ↴ avatars (method) dnajs.Profile.avatars(...avatarRec... ⓘ
7   ↴ delete
8   ↴ edit
9   ↴ lolopi
10  ↴ personalData
11  ↴ Exec pluckCreationDate
12  ↴ pluckHumanReadableProfileName
13  ↴ pluckIsAutoCreatedProfile
14  ↴ pluckIsAutoStartEnabled
15  ↴ pluckIsDefaultKidsProfile
16  ↴ pluckIsFirstUse
17  ↴ pluckIsInstantQueueEnabled
```

REPL

Results



Queries:





# Develop in container

Develop in a prod-like environment

Mount the application folder into the local image

\* Exclude node\_modules !

Run nodemon within the container to watch for changes



# Debug in container

Debug in a prod-like environment

Expose the node.js debug port from  
the container



# Microservices



Auth (Zuul)



BFF

BFF

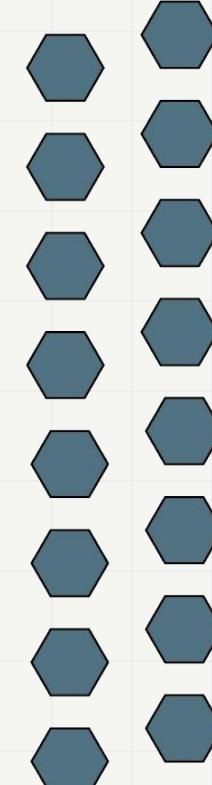
BFF

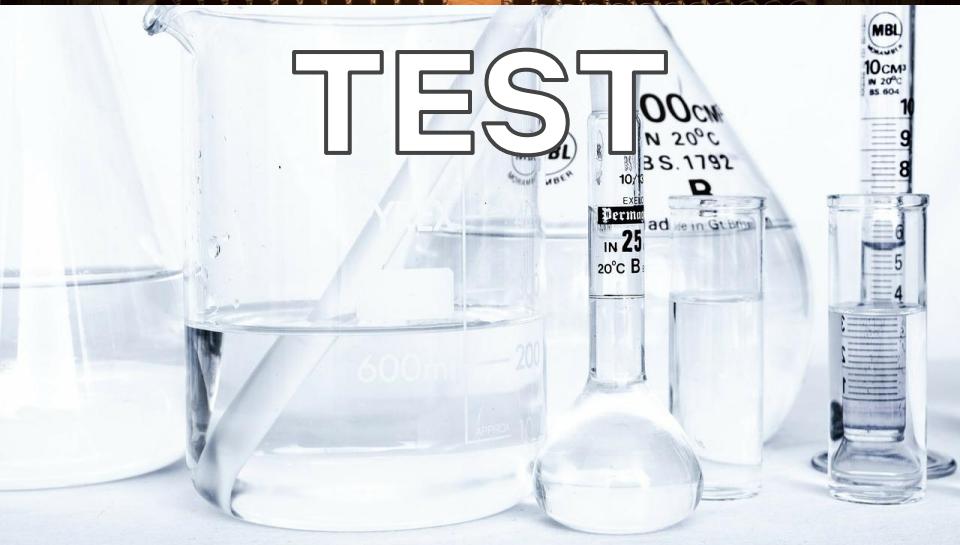
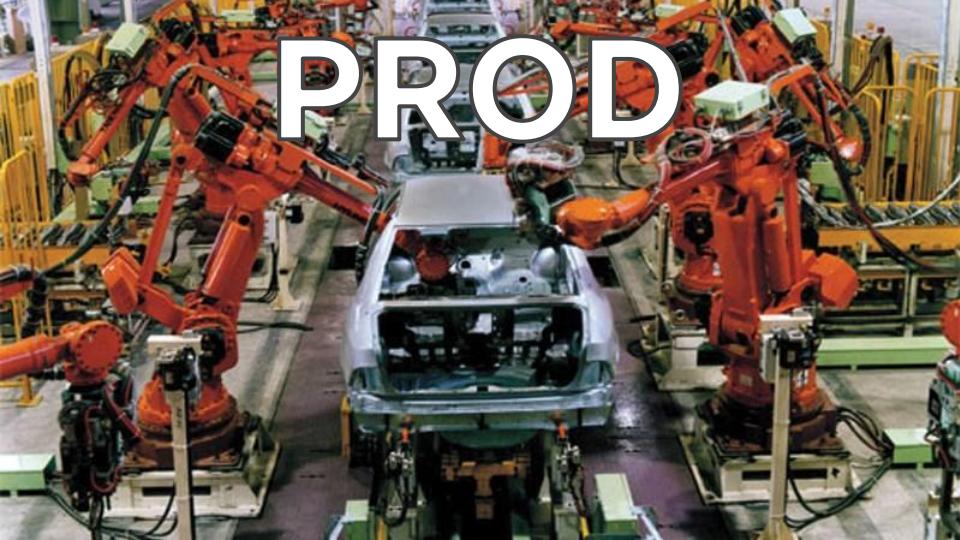
BFF

Playback

DNA

Midtier





# Test in container

Unit tests:

- Rest route handler functions directly

Integration tests:

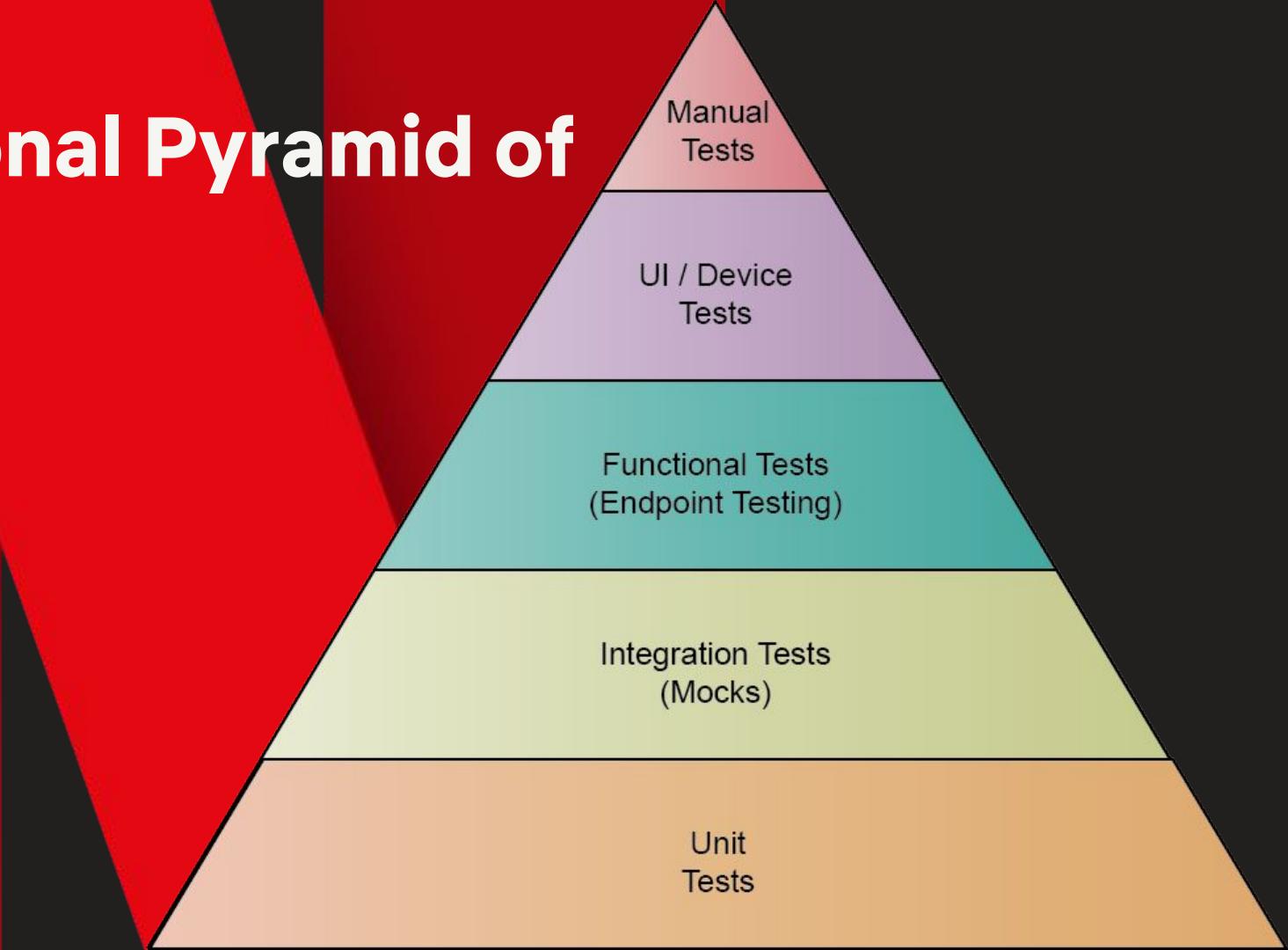
- Test in-process with mocked platform components

Functional tests:

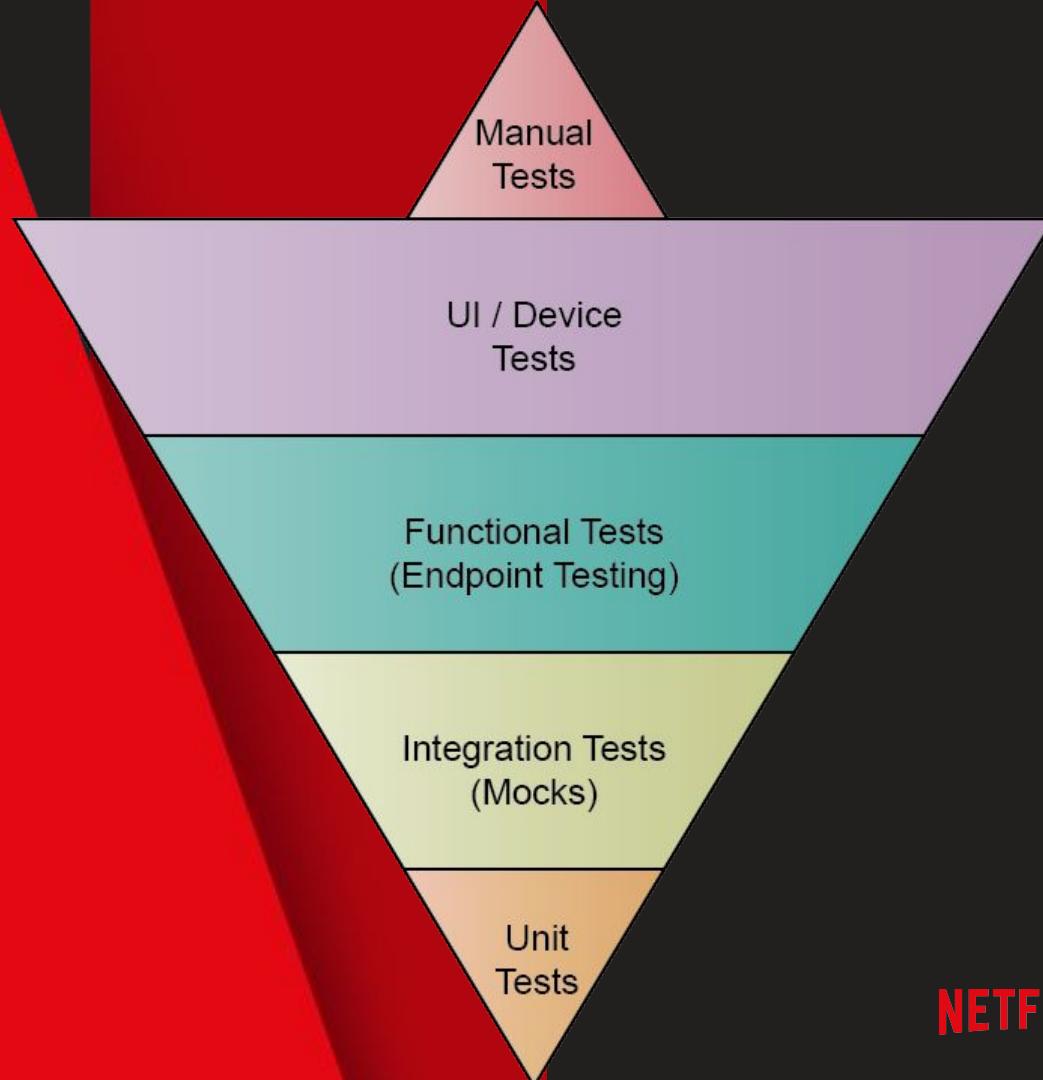
- Black-box testing of the exposed endpoints



# Traditional Pyramid of Testing

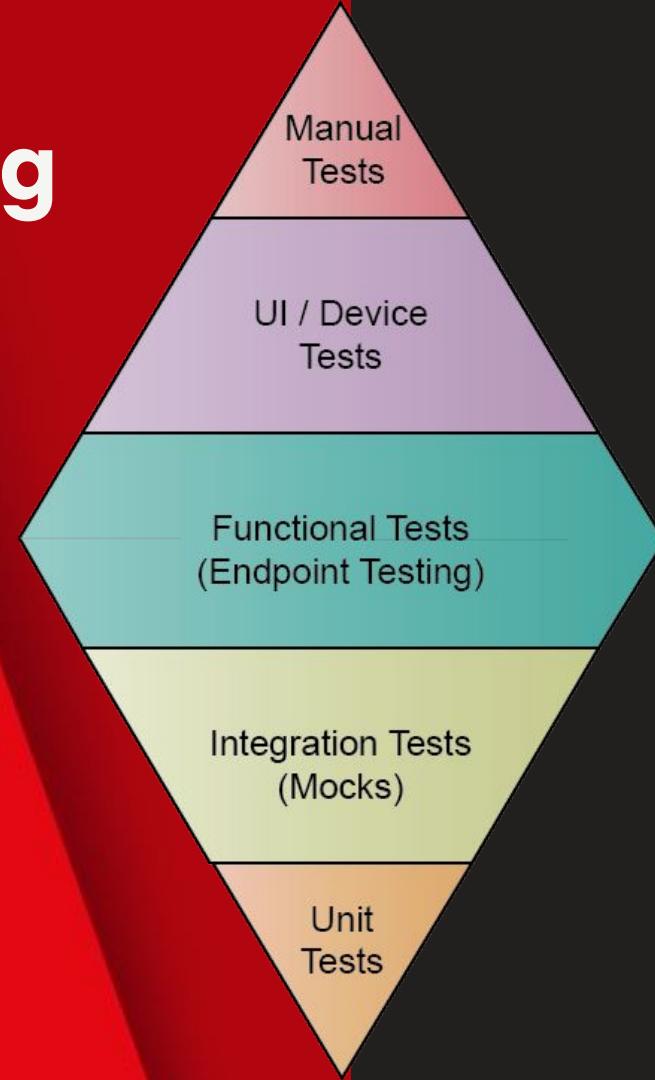


# Inverted Pyramid of Testing



NETFLIX

# Diamond of Testing



NETFLIX

# Conclusion

BFFs aligned with frontend teams enable:

- Evolve the front and backends together, quickly
- Fluidity in deciding where functionality lives

However...

- Increased complexity
- Require staffing to develop, rollout, and maintain

Identify which pieces work best for your use case



# Thank you

@brianleathem



# Endpoint testing

