Optimizing Data Access

For Frontend Use Cases
With A Modular Api Gateway

NETFLIX

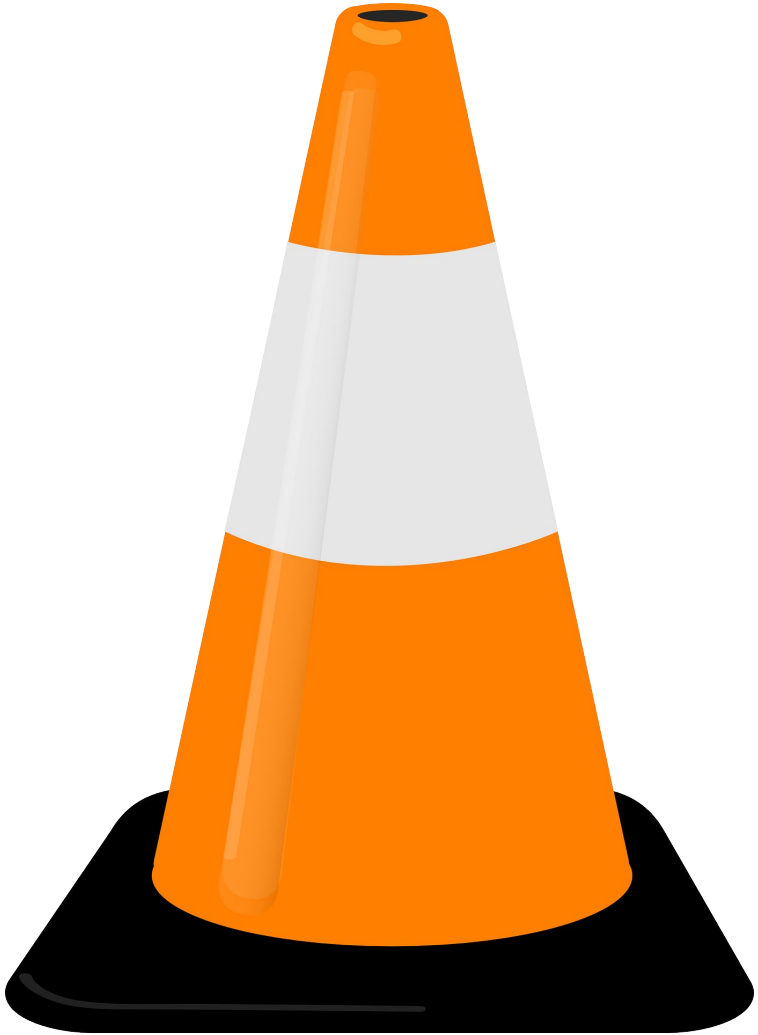

Naming things is hard!

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".

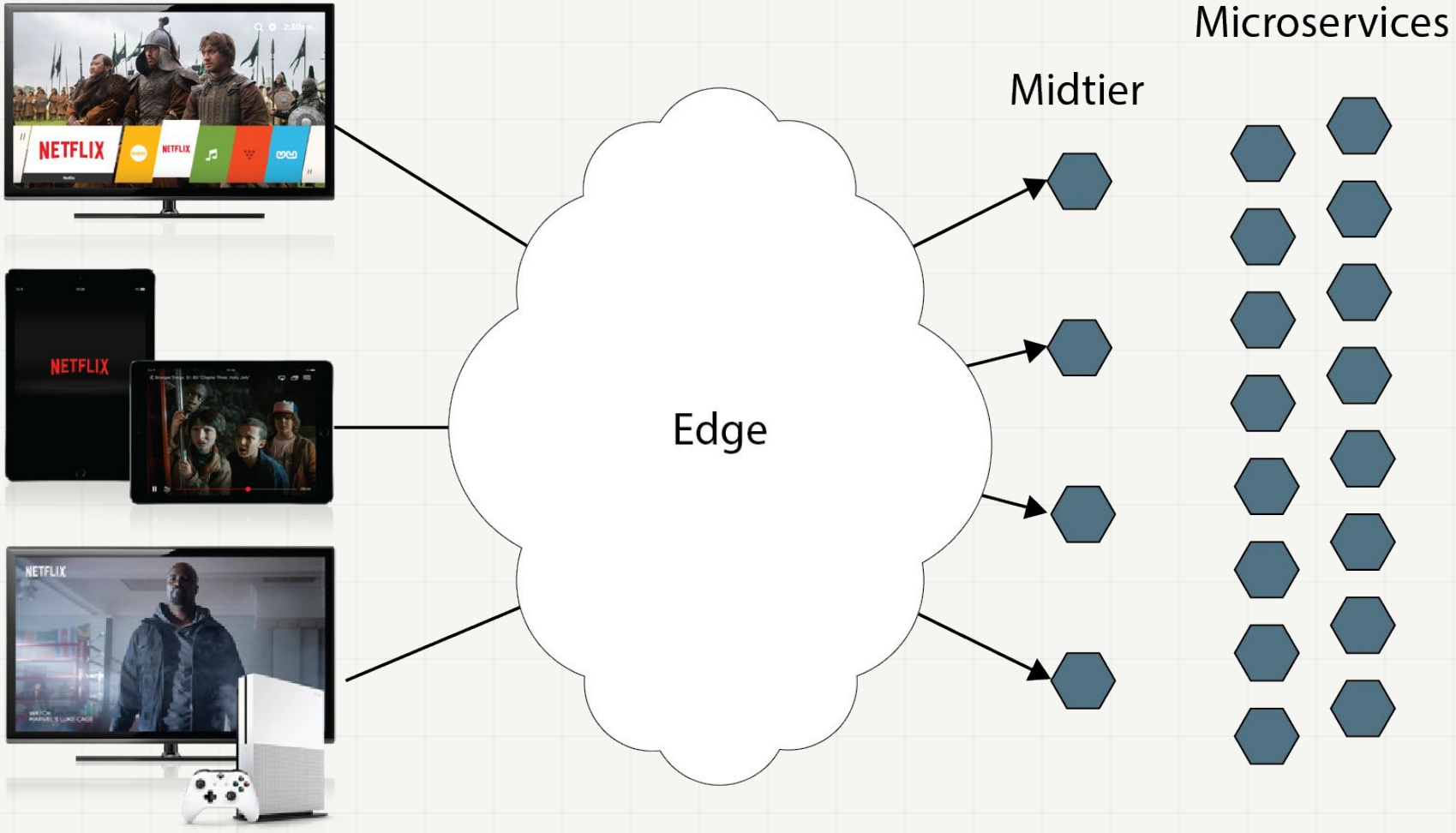


Who am I?

Brian Leathem

Senior Software Engineer @ Netflix
- Edge Developer Productivity

@brianleathem


Microservices
Midtier
Edge
NETFLIX
NETFLIX
NETFLIX
NETFLIX

The Ideal Developer Experience

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

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

@ Scale

Brian Leathem - Optimizing Data Access
All things Open 2018

Traffic

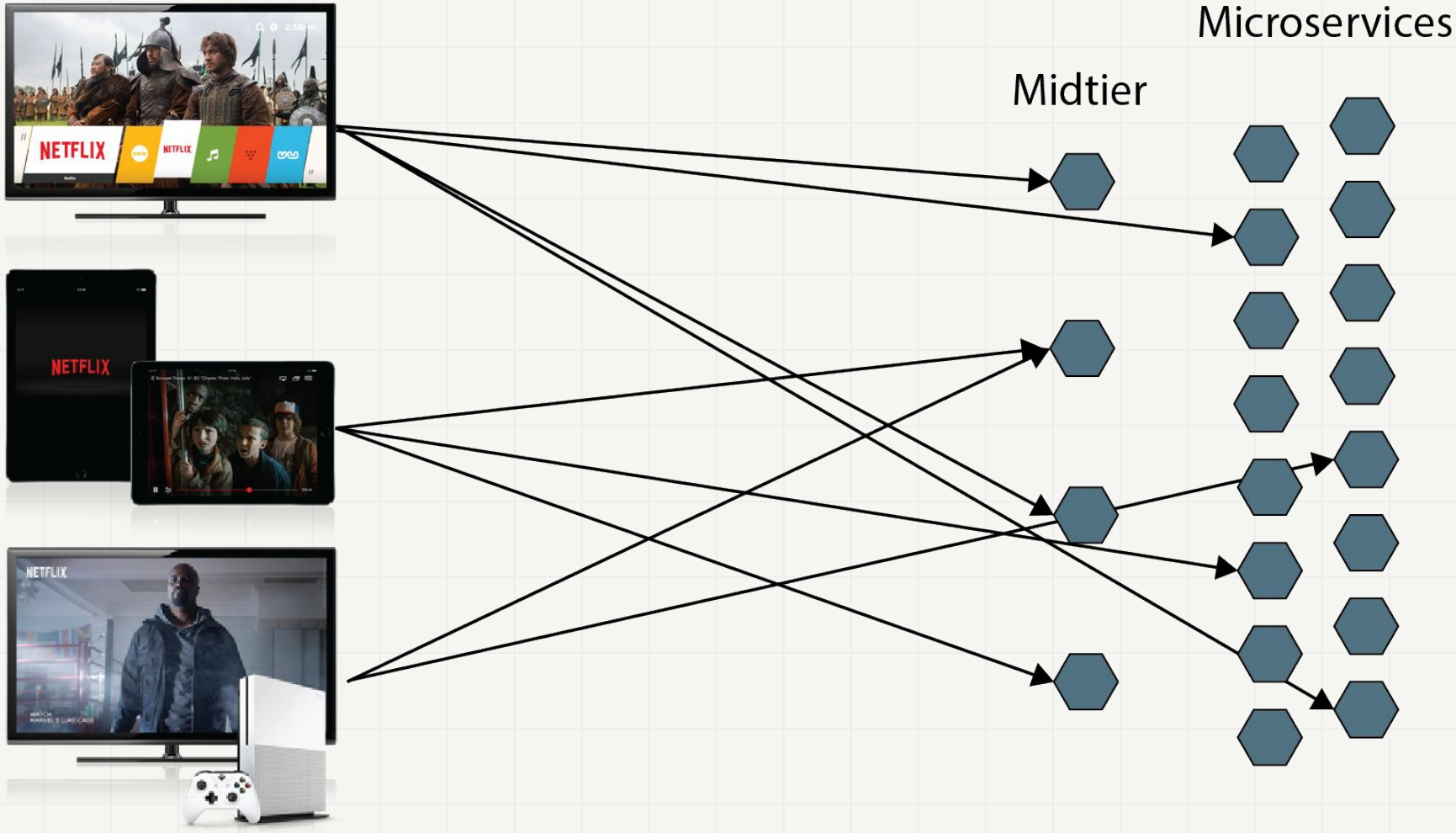

A 54 Northwich, Chester
Middlewich, Holmes Chapel
NORTH WALES, Preston,
Manchester, Leeds M6
18
M6
A
276.0


NETFLIX
NETFLIX
MARVEL
LUKE CAGE
NETFLIX
NARCOS
NETFLIX
STRANGER THINGS

Microservices


Teams

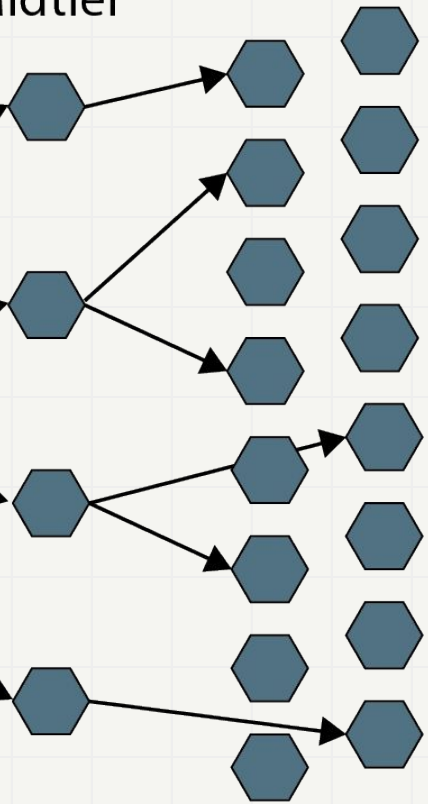
Backend For Frontend (BFF)

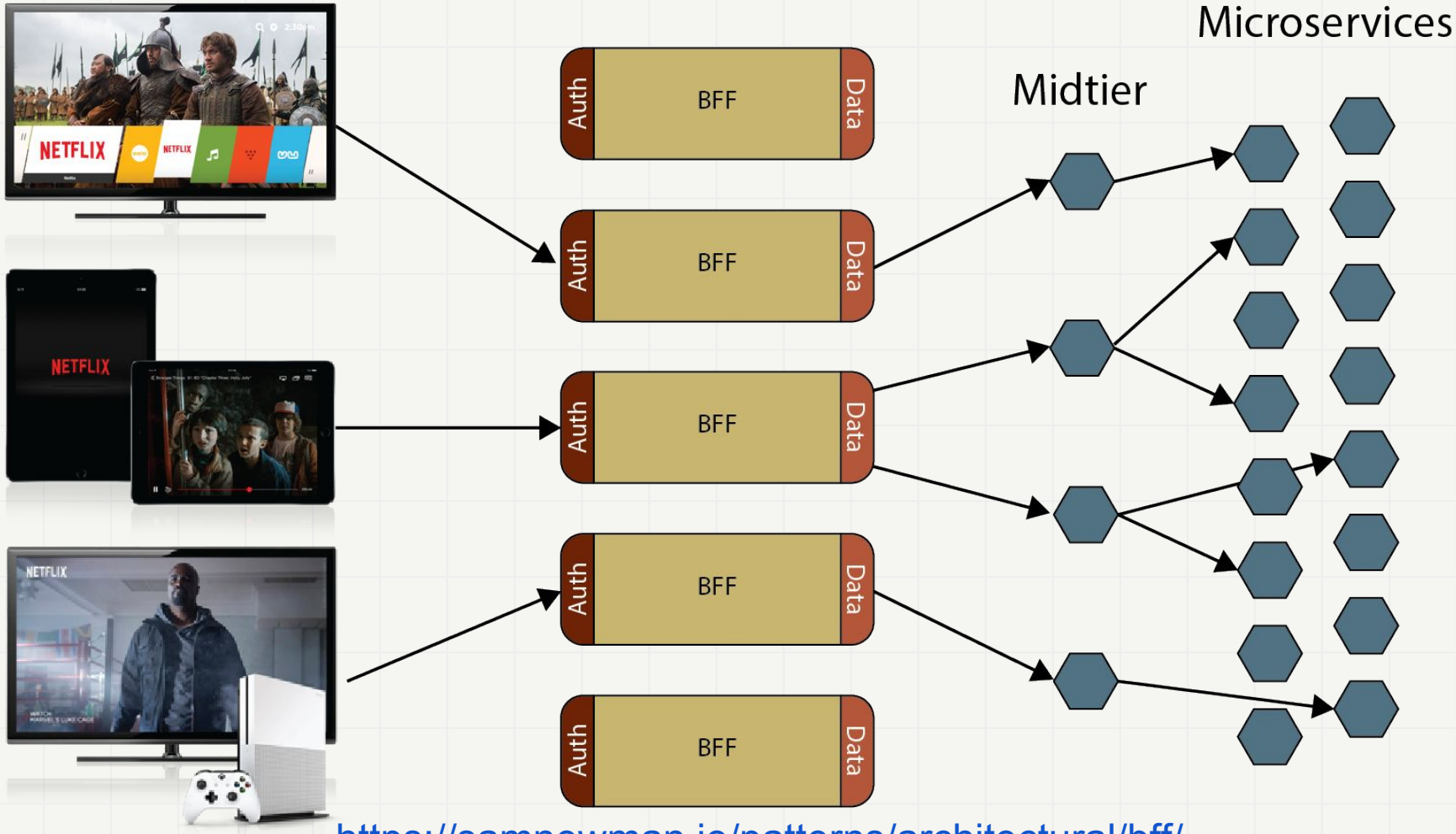

Microservices
Midtier
NETFLIX
NETFLIX
NETFLIX
NETFLIX

General Purpose API Gateway

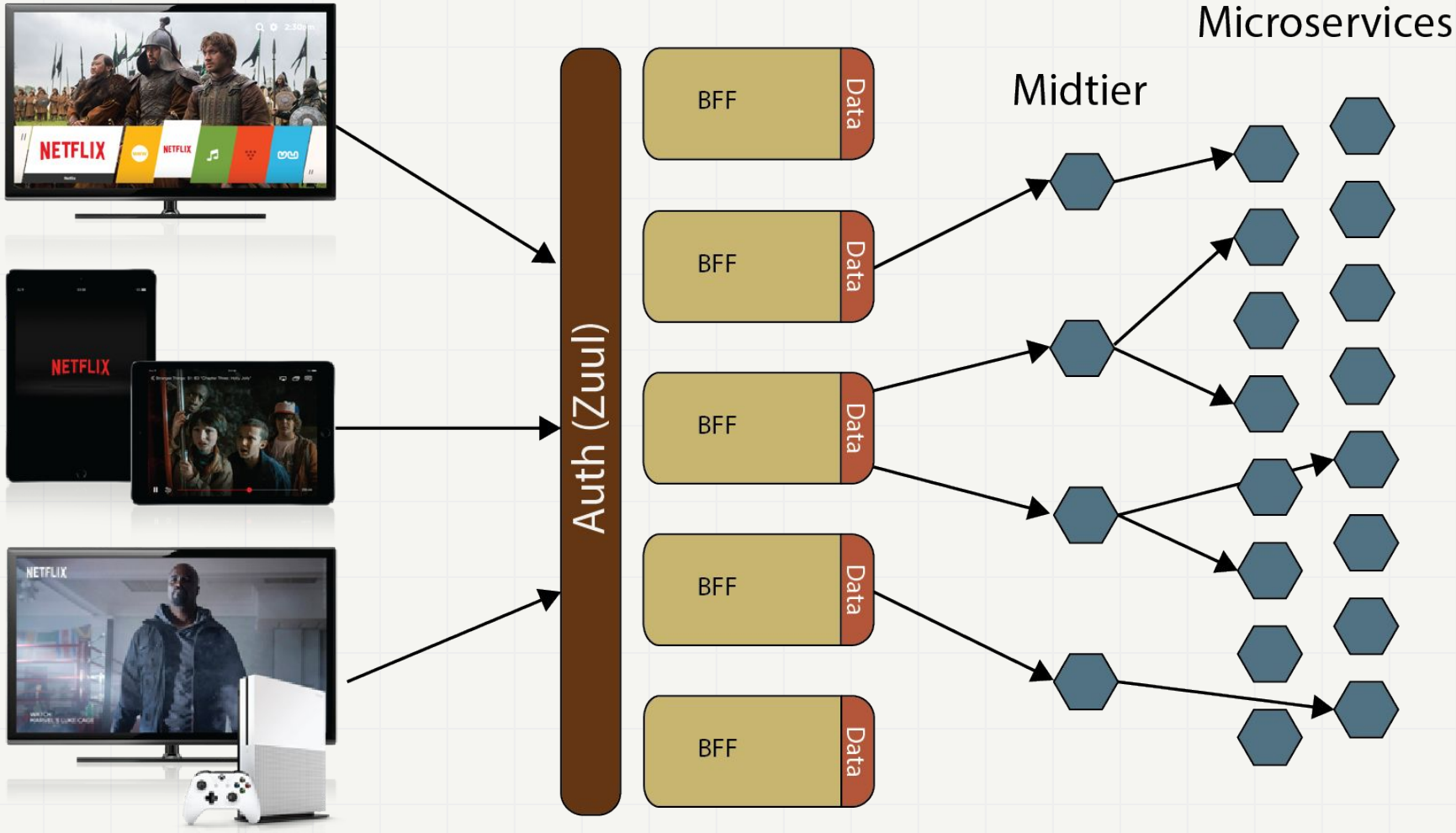
Microservices

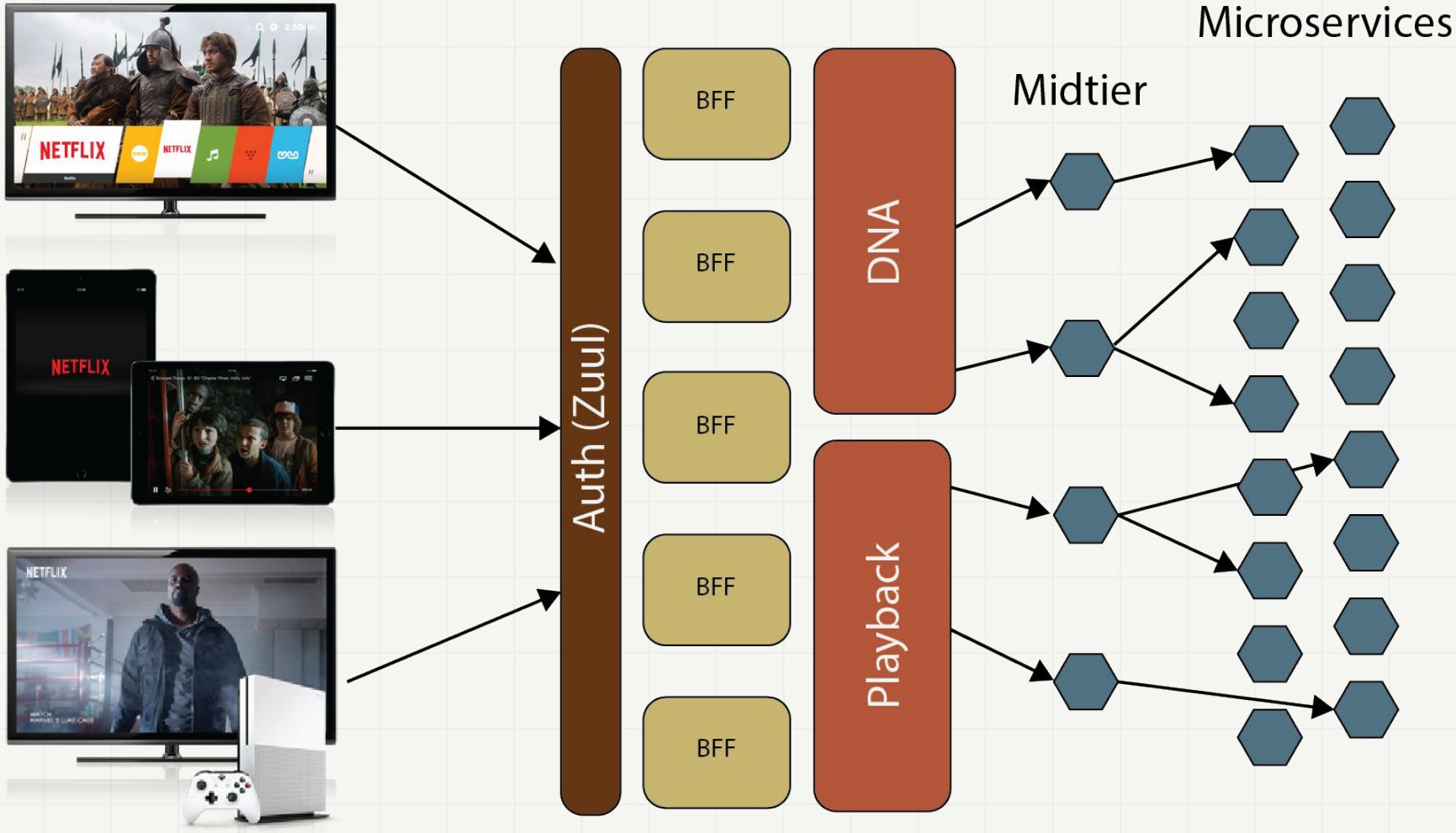
Midtier





https://samnewman.io/patterns/architectural/bff/


Microservices
Midtier
NETFLIX
NETFLIX
NETFLIX
NETFLIX
Auth (Zuul)
BFF
Data
BFF
Data
BFF
Data
BFF
Data
BFF
Data


Microservices
Midtier
NETFLIX
NETFLIX
NETFLIX
NETFLIX
Auth (Zuul)
BFF
BFF
BFF
BFF
BFF
DNA
Playback

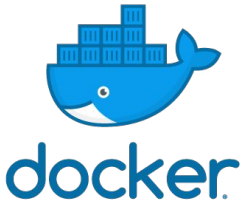
How we Built our BFF

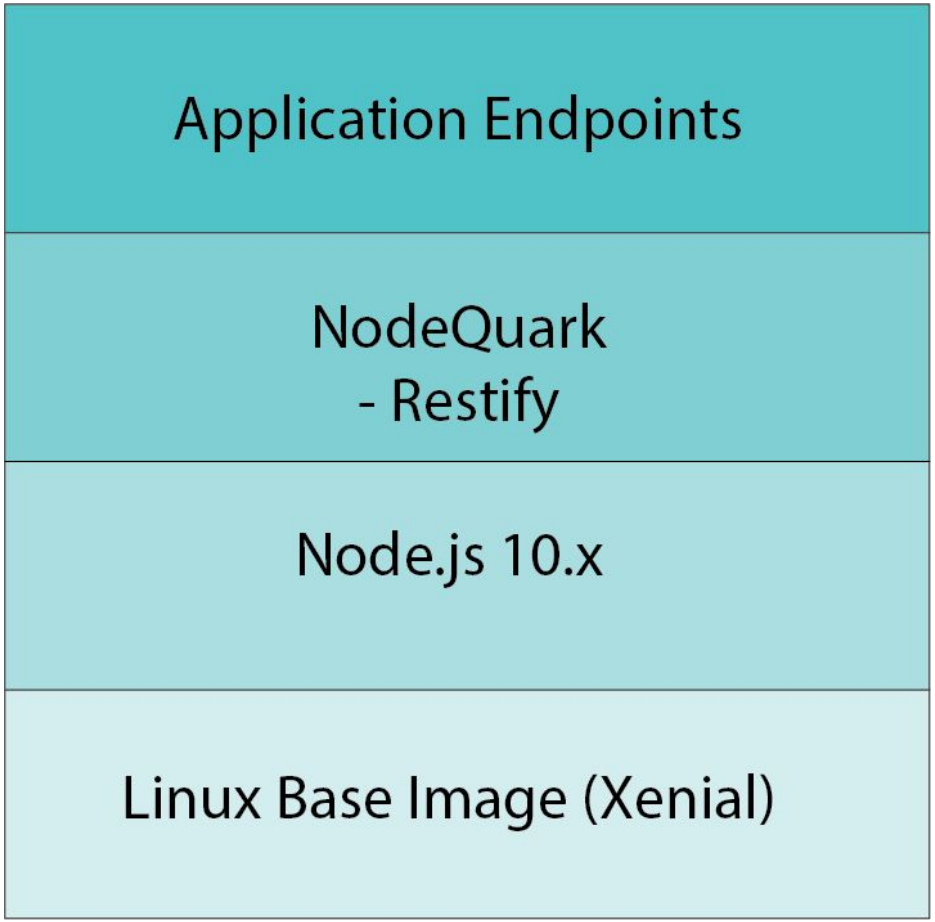
Brian Leathem - Optimizing Data Access
All things Open 2018


docker

Inside the container

- Node.js
- Restify for HTTP
- Eureka for service registration and discovery
- Archaius for configuration management
- Atlas for metrics





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.

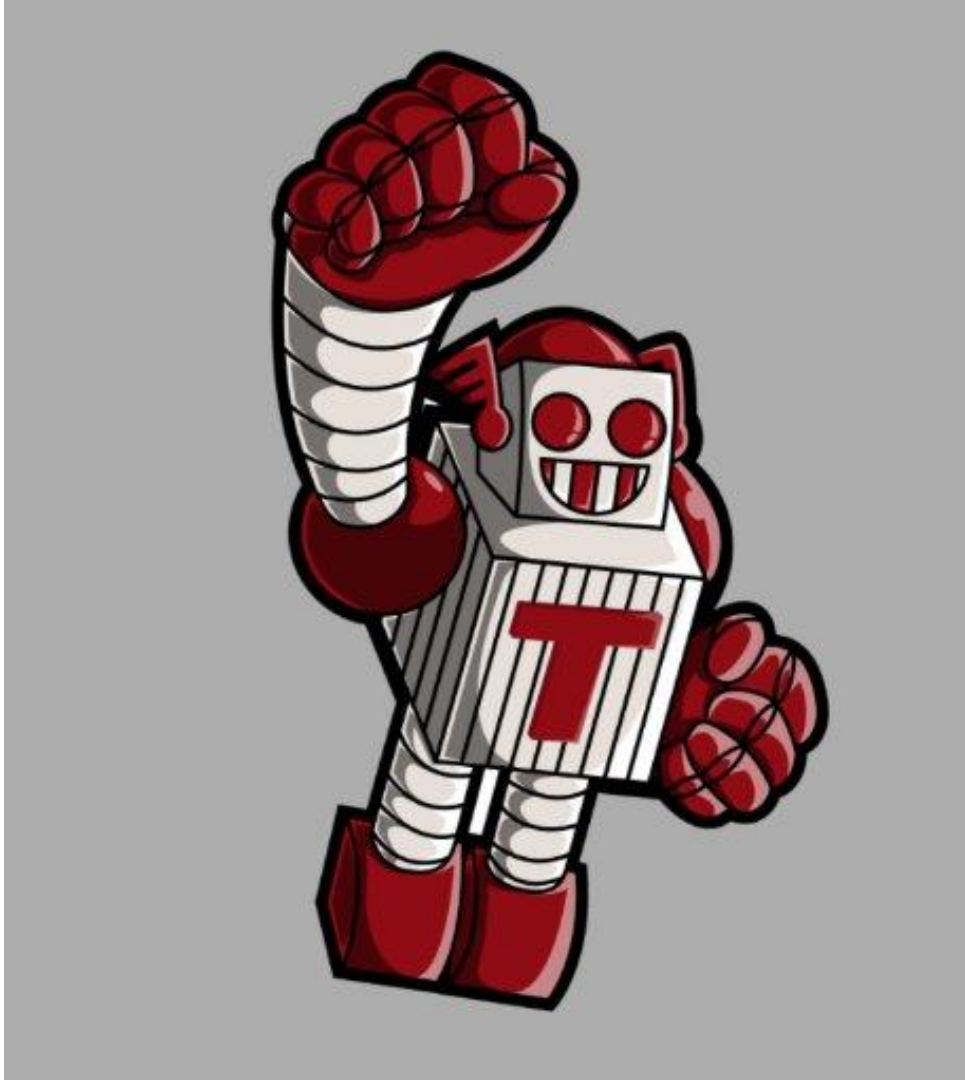
Falcor: distributed JSON graph

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

Scaling traffic with Titus

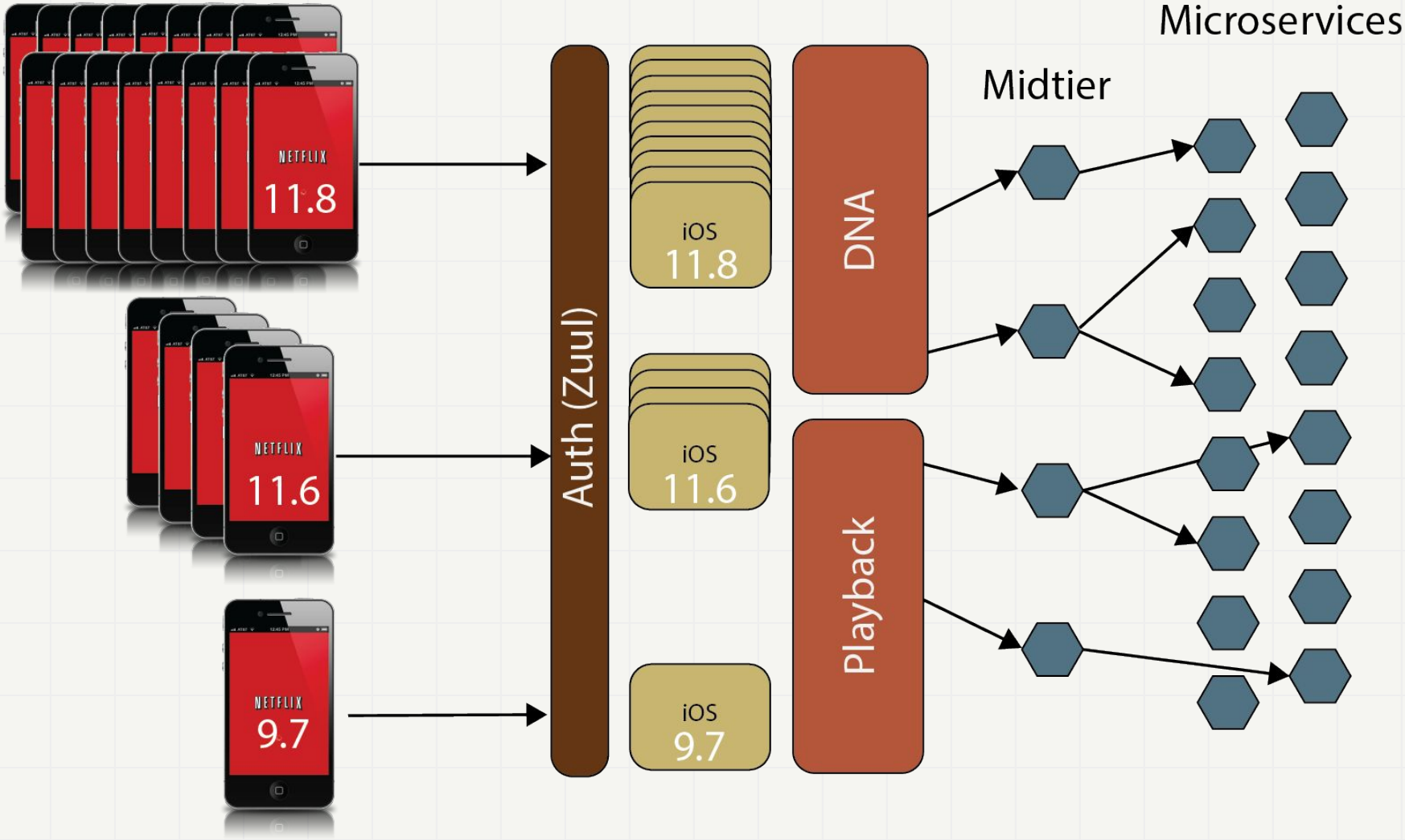
- A production ready container platform
- Integration with AWS
- Netflix OSS integration

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



Dynamic Routing

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

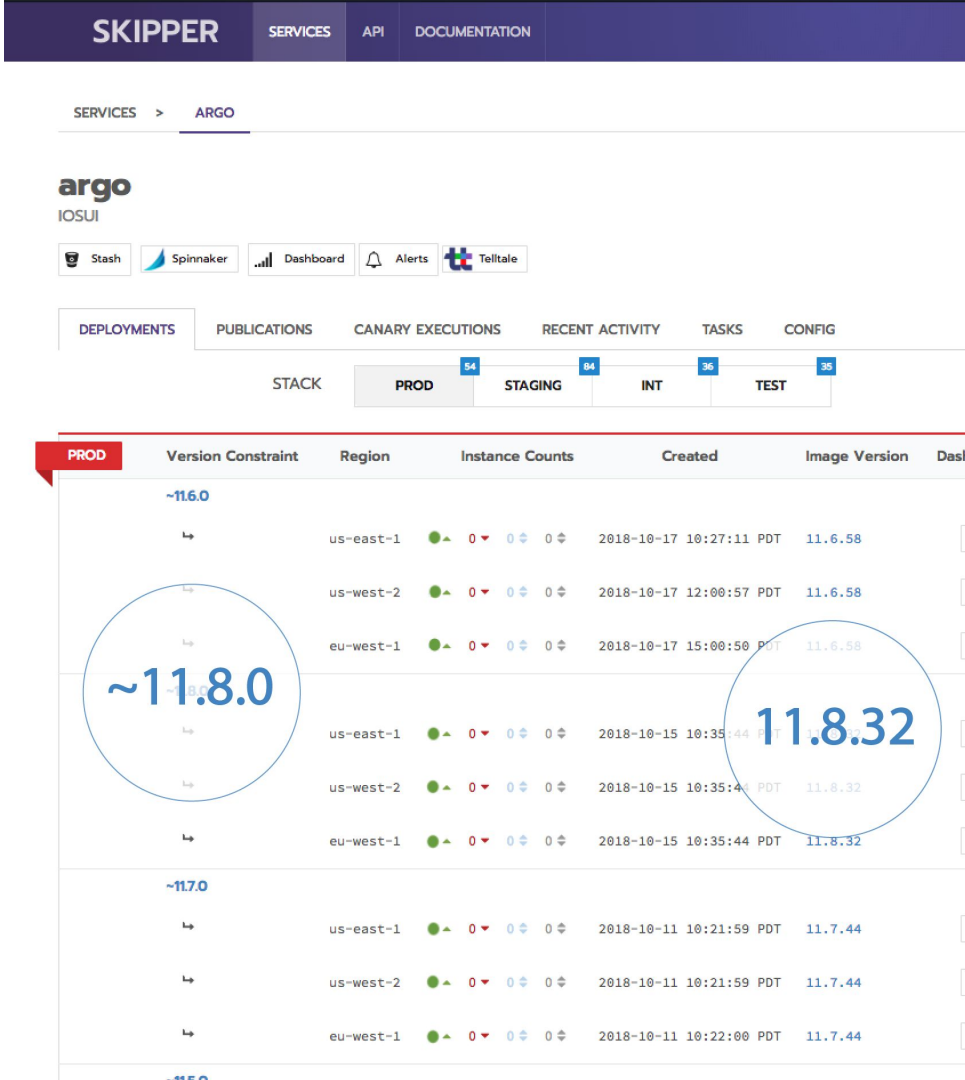

Microservices
Midtier
NETFLIX
11.8
NETFLIX
11.6
NETFLIX
9.7
Auth (Zuul)
iOS
11.8
iOS
11.6
iOS
9.7
DNA
Playback

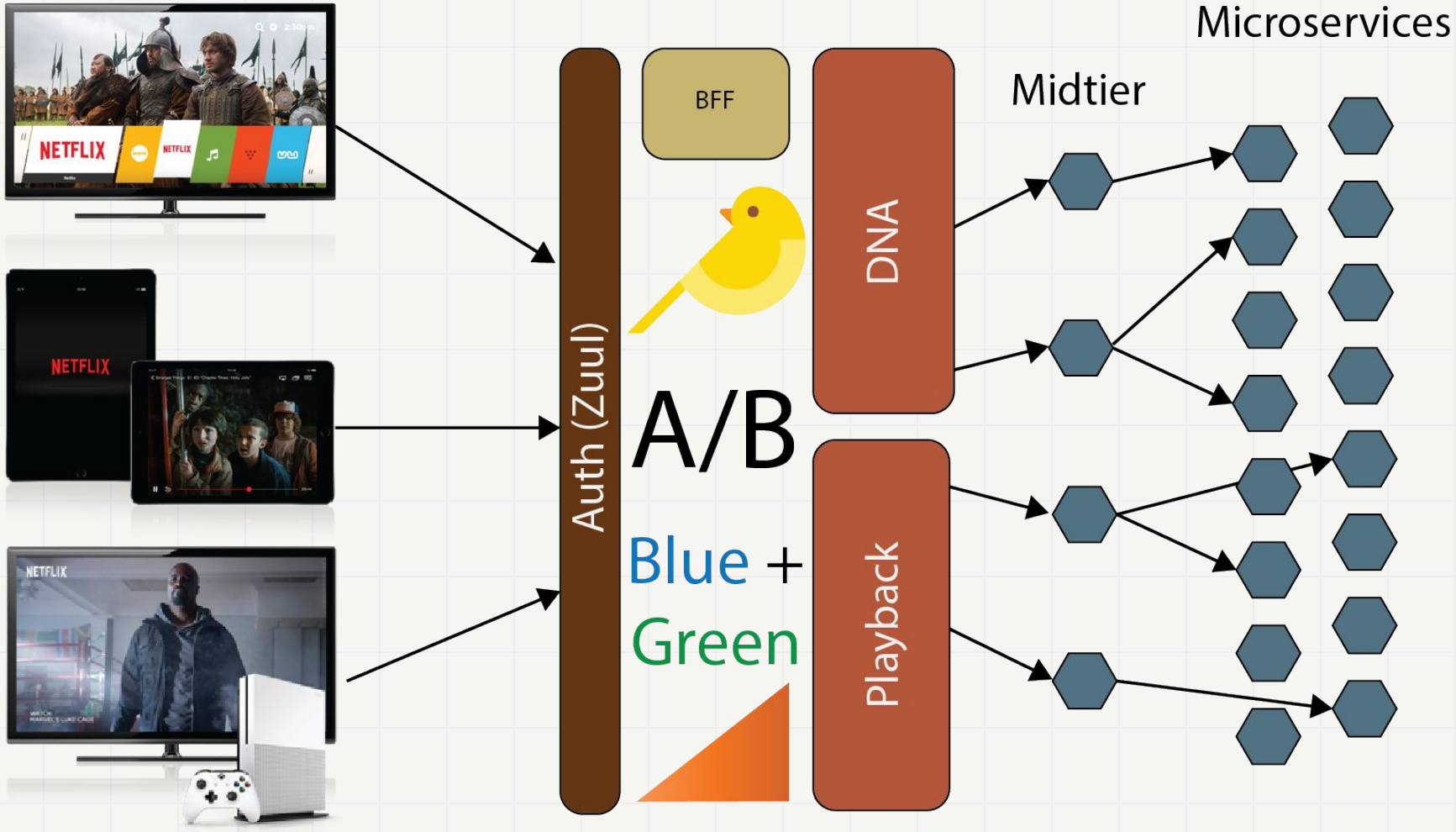
Scaling Versions

Multiple deployed versions of a BFF endpoint to support

Client has a hardcoded endpoint

Deploy patch releases to a given endpoint




Microservices
Midtier
BFF
Auth (Zuul)
A/B
Blue +
Green
DNA
Playback
NETFLIX

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

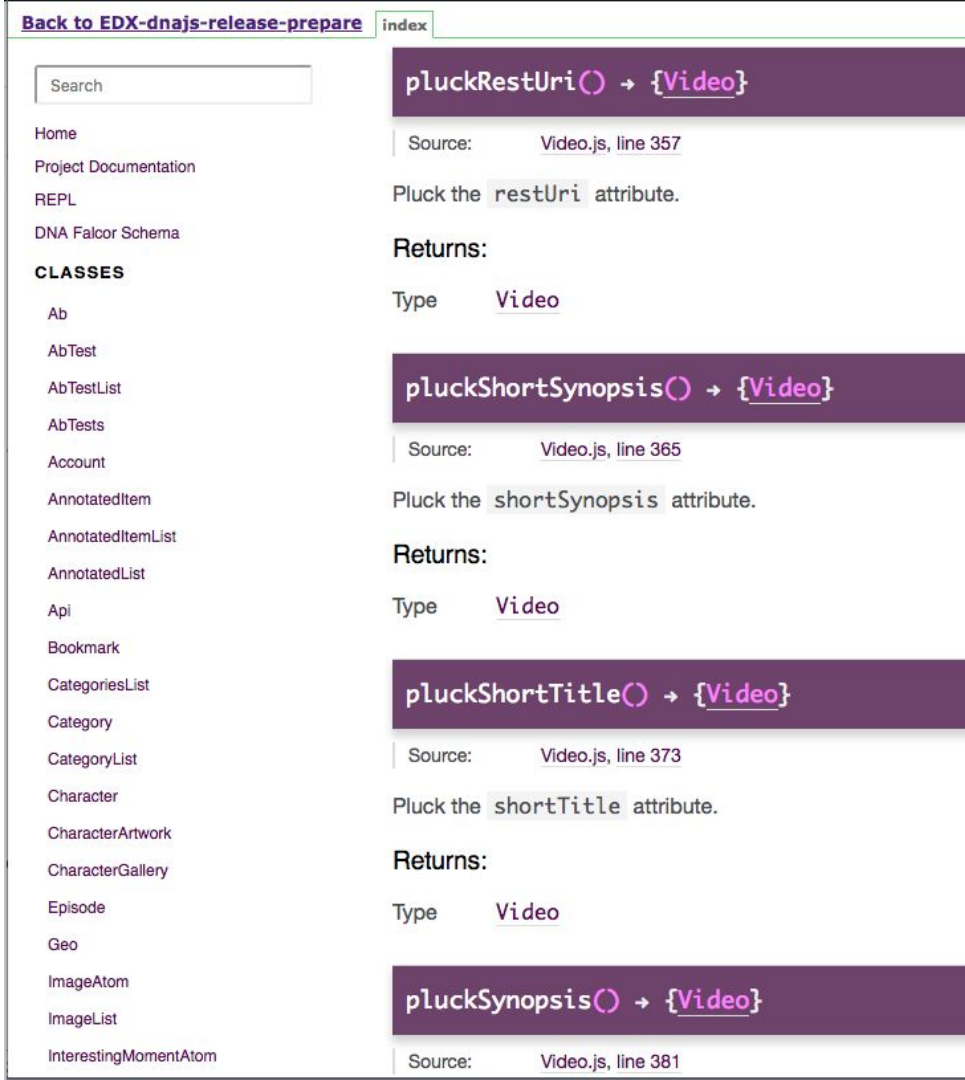
Map client requests into backend requests

From the client:

- Path requests from a Falcor client

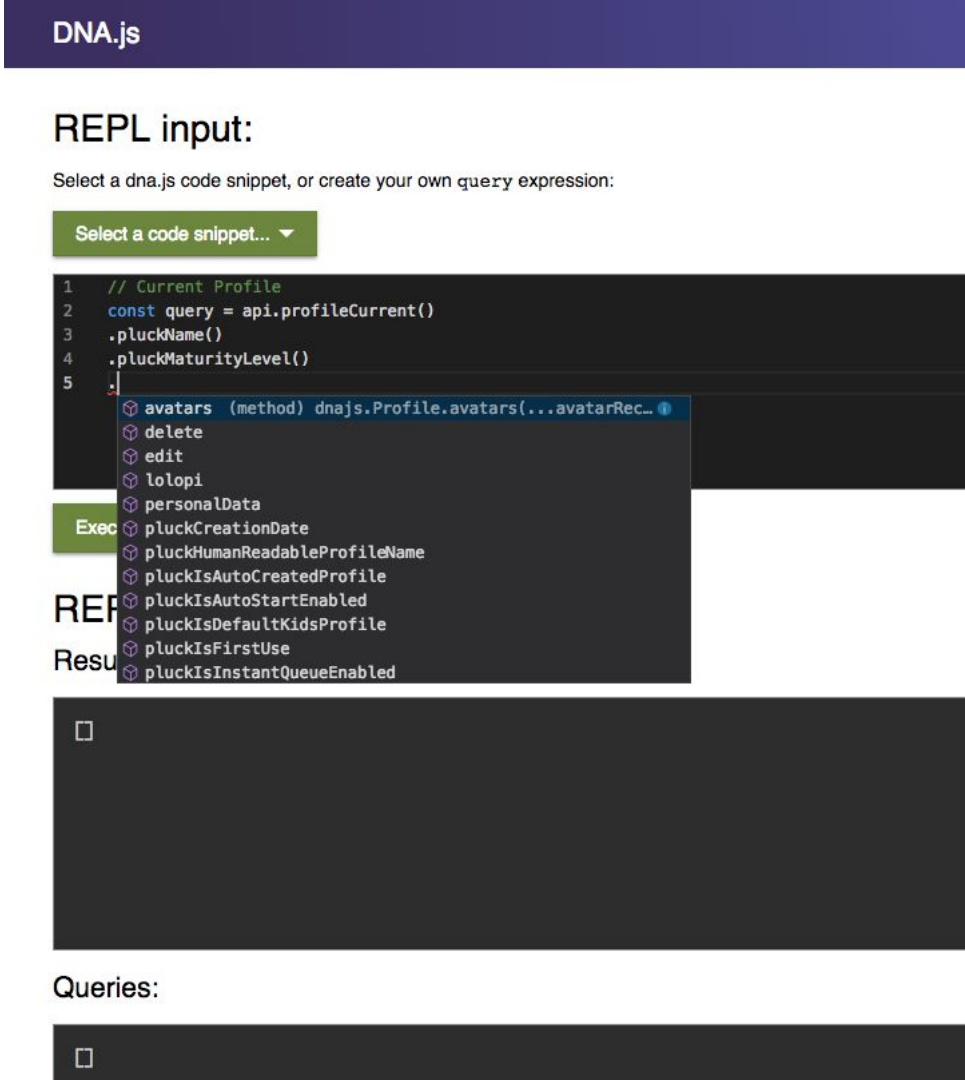
From the backend:

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



Data Discovery: Online REPL

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



Develop in container

Develop in a prod-like environment

Mount the application folder into the local image

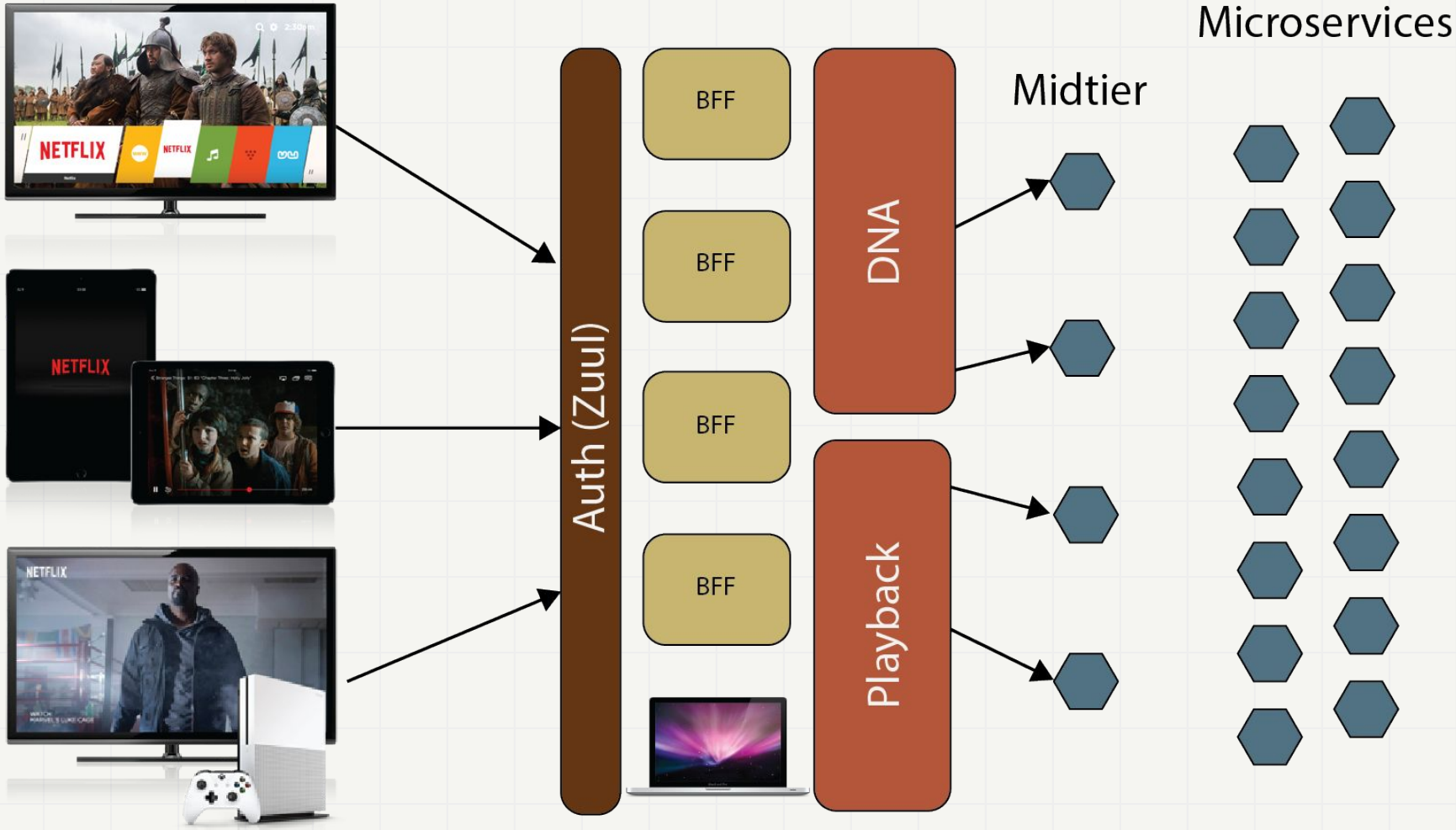
* Exclude node_modules !

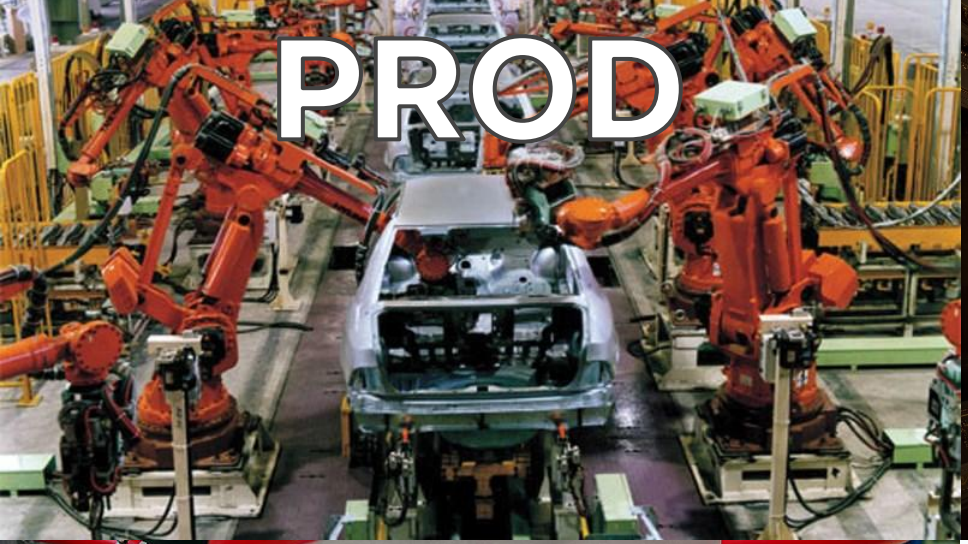
Run nodemon within the container to watch for changes

Debug in container

Debug in a prod-like environment

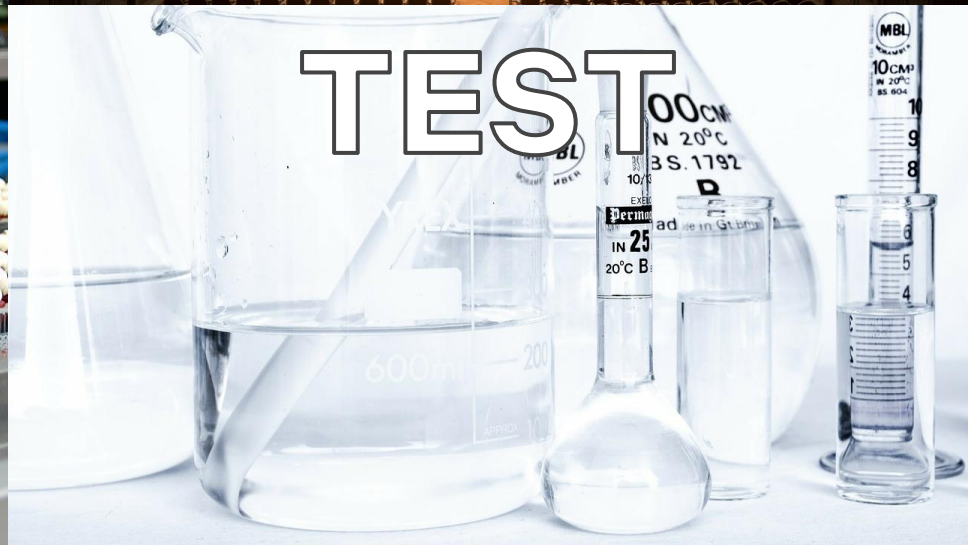
Expose the node.js debug port from the container


Microservices
Midtier
NETFLIX
NETFLIX
NETFLIX
NETFLIX
Auth (Zuul)
BFF
BFF
BFF
BFF
DNA
Playback


PROD


STAGING


INT


TEST

Test in container

Unit tests:

- Rest route handler functions directly

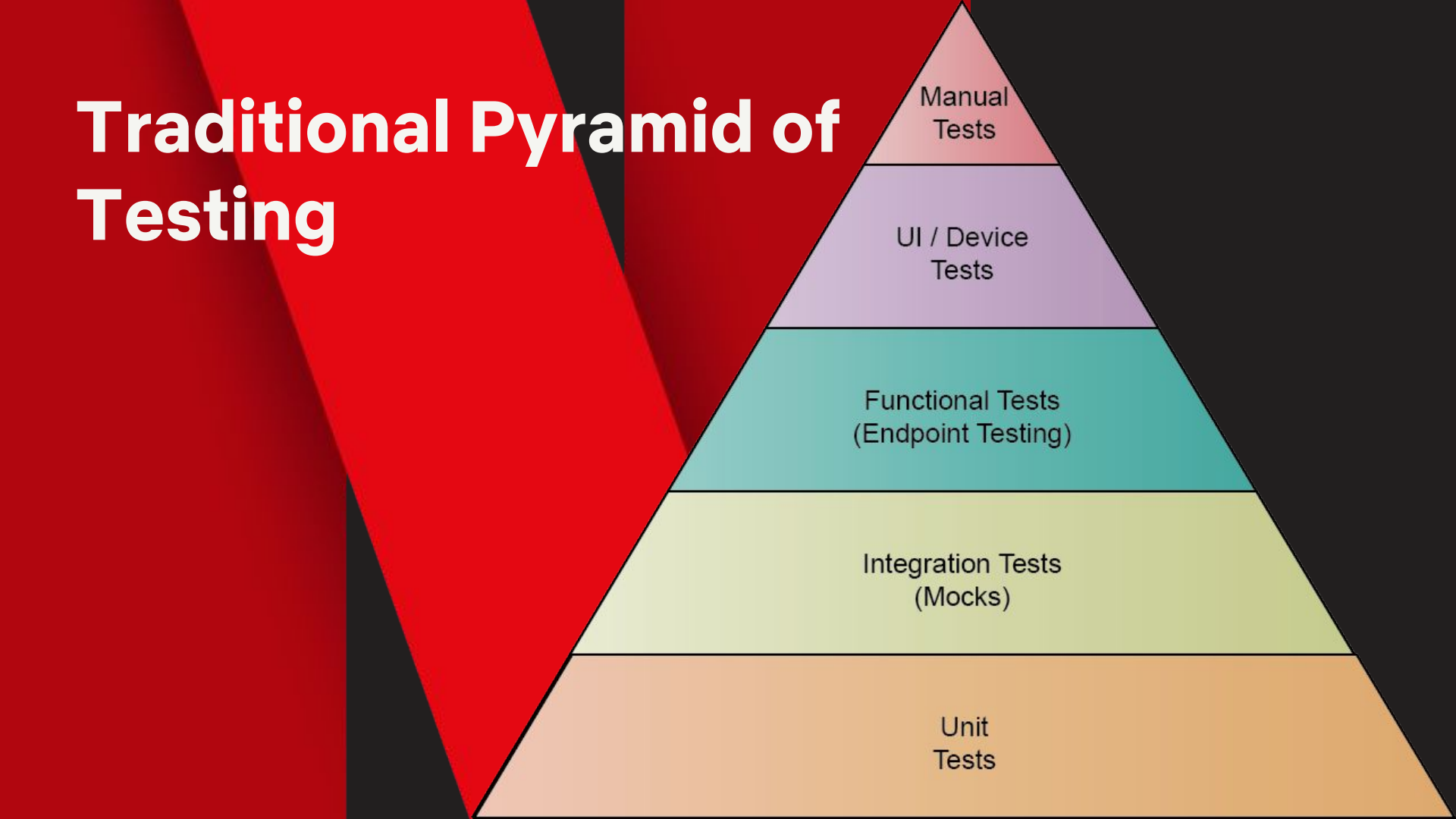
Integration tests:

- Test in-process with mocked platform components

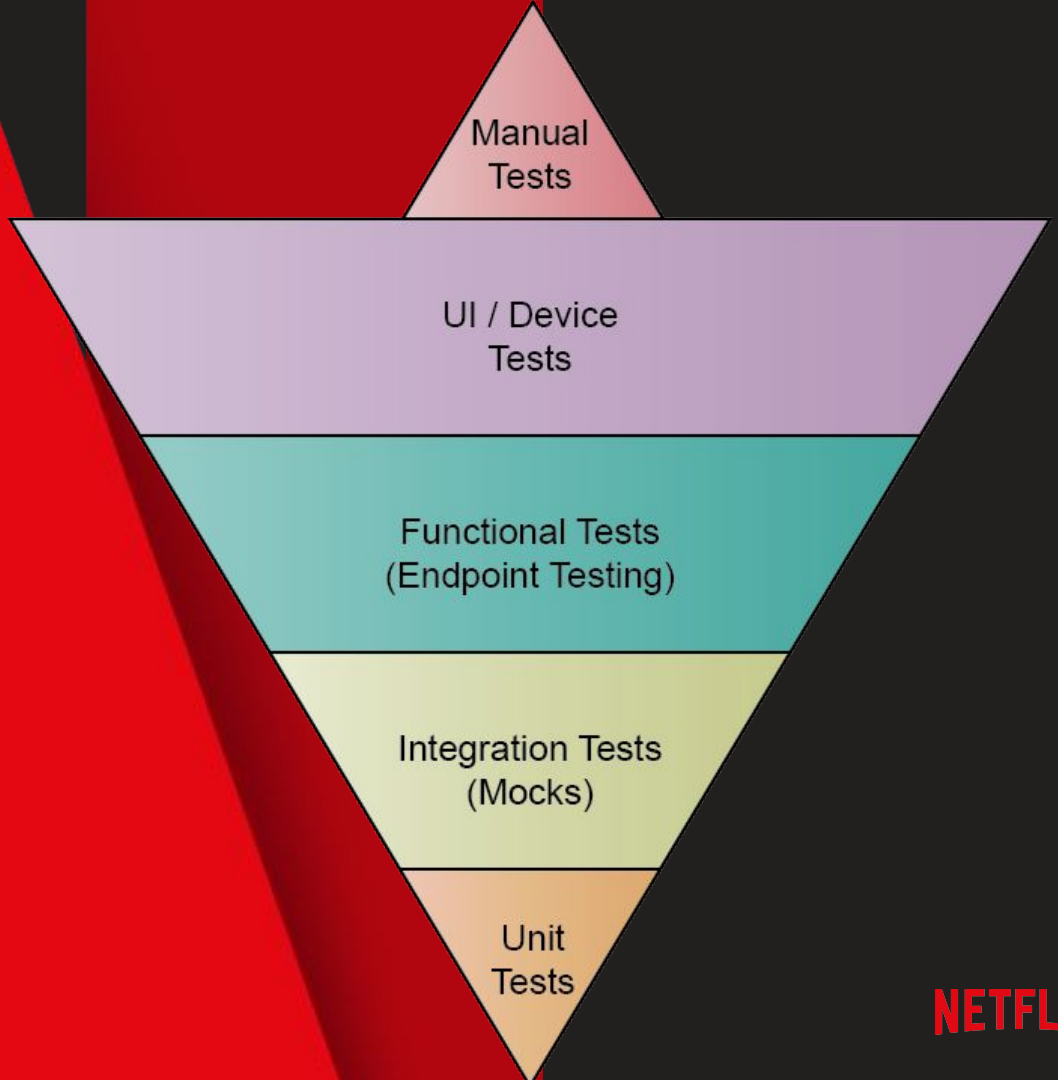
Functional tests:

- Black-box testing of the exposed endpoints

Traditional Pyramid of Testing

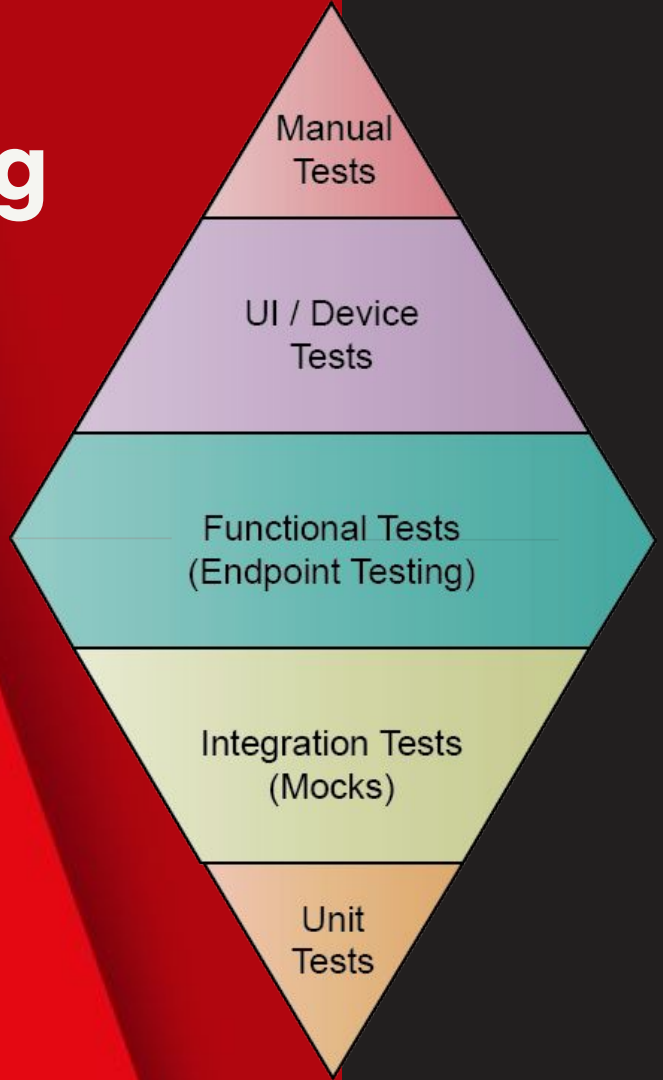


Inverted Pyramid of Testing



NETFLIX

Diamond of Testing



NETFLIX

Conclusion

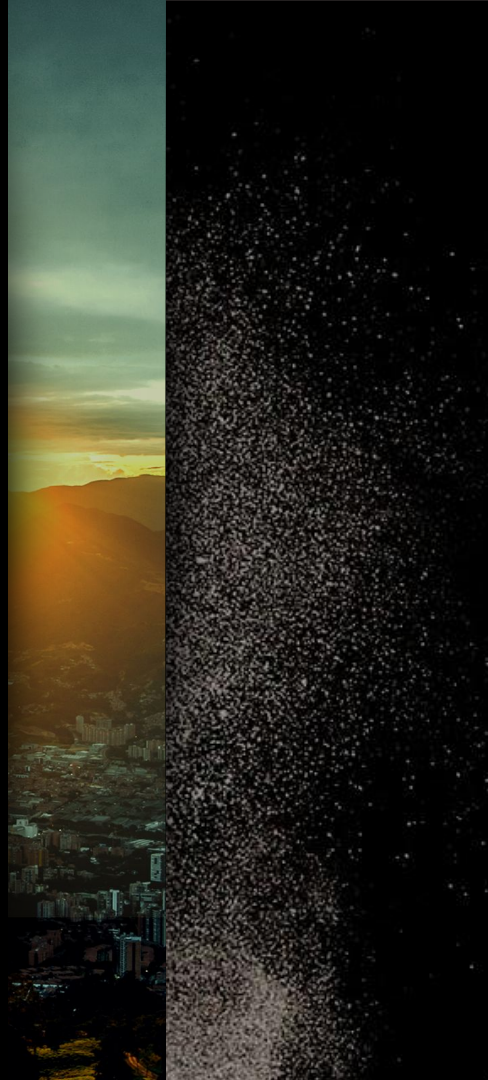
BFFs aligned with frontend teams enable:

- 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

NETFLIX