Maintaining Backwards Compatibility

Mark Heath
CLOUD ARCHITECT

@mark_heath www.markheath.net

Overview

Breaking changes

Importance of versioning APIs

- Versioning strategies

Backwards compatibility

- Support old and new clients

Non-breaking Changes

Not every change is a "breaking change"

Adding a new endpoint

- e.g. /api/special-offers

Query string parameters

- /api/events?category=musical
- /api/events?category=musical&fromDate=2020-08-01&toDate=2020-09-01

Modifying DTOs

```csharp
public class Event
{
   public Guid EventId { get; set; }
   public string Name { get; set; }
   public int Price { get; set; }
   public string Artist { get; set; }
   public DateTime Date { get; set; }
   public string Description { get; set; }
   // new property:
   public string Location { get; set; }
   // ...
}
```

```json
{
   "eventId": "2db9c8f0-e865-4bca-b389-
03b09a9fdabf",
   "name": "John Egbert Live",
   "price": 65,
   "location": "Edmonton Hall",
   // ...
}
```

JSON parsers usually ignore unexpected fields

Breaking Changes to Values

Original definition:

```
public enum EventStatus
{
   OpenForBooking,
   SoldOut
}
```

Updated definition:

```
public enum EventStatus
{
   OpenForBooking,
   SoldOut,
   Cancelled
}
```

What will v1 clients do if they receive an EventStatus of Cancelled?

Replacing Properties

Original definition:

```
public class Event
{
   public Guid EventId { get; set; }
   public string Name { get; set; }
   public int Price { get; set; }
   public string Artist { get; set; }
   public DateTime Date { get; set; }
   // we can only show one image:
   public string ImageUrl { get; set; }
   // ...
}
```

Updated definition:

```
public class Event
{
   public Guid EventId { get; set; }
   public string Name { get; set; }
   public int Price { get; set; }
   public string Artist { get; set; }
   public DateTime Date { get; set; }
   // support multiple images:
   public string[] ImageUrls { get; set; }
   // ...
}
```

V1 clients rely on the ImageUrl property

What Changes Are “Safe”?

Additive changes are generally safe

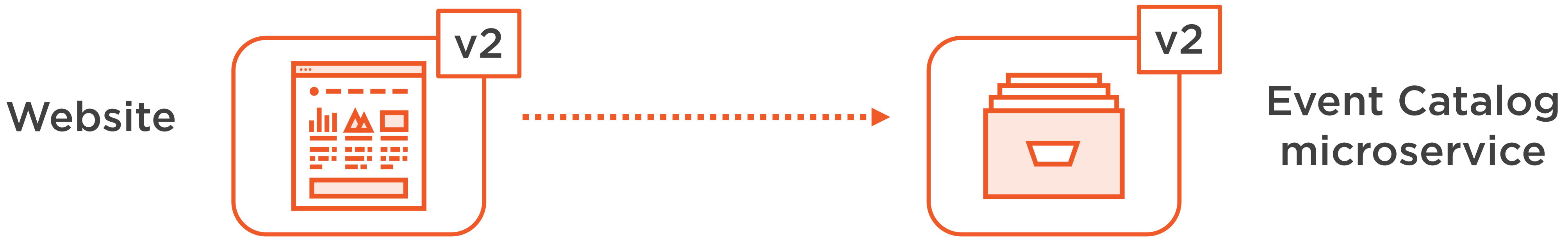
- Adding new endpoints
- Adding new (optional) query string parameters
- Adding new properties to DTOs

Replacing or removing things cause breaking changes

- Renaming a DTO property or endpoint
- Removing a DTO property or endpoint
- Changing the type of a DTO property

A Simple but Dangerous Solution

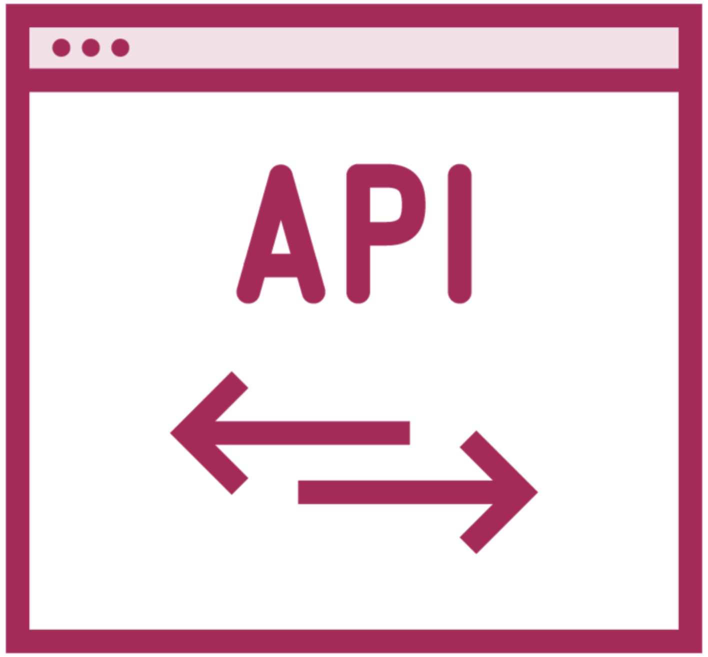
Why not simply upgrade all clients whenever we making a breaking change?



1	**Microservices should be autonomous**	Rolling upgrades
2	**You cannot control all clients**	e.g. Mobile applications
3	**Owned by independent teams**	Independent release schedules
4	**Are you aware of all clients?**	e.g. Report generators

Once published, treat your APIs as immutable

Make changes by publishing a new version of your API



Sometimes we need to make breaking changes

Maintain backwards compatibility

- Older clients can still call the API
- They can upgrade later to use the new version

We need a way to version APIs

- Many possible techniques
- No agreed-upon standard

Maintain backwards compatibility for older clients

We need to version our APIs

Versioning APIs

1 **Path**

https://localhost:5001/api/events

https://localhost:5001/api/v1/events

https://www.googleapis.com/drive/v3/files

2 **Query string**

https://localhost:5001/api/events?version=1.3

3 **HTTP header**

X-Version: 1.3

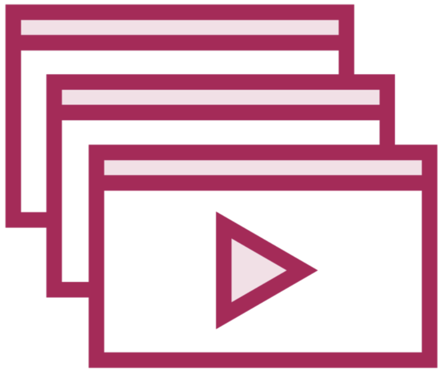
RESTful Versioning

REST APIs are based on "resources"

Make use of HTTP methods (e.g. GET, POST, PUT)

Can use custom vendor media types Accept and Content-Type headers for versioning

e.g. Accept: application/vnd.globoticket.event.v3+json

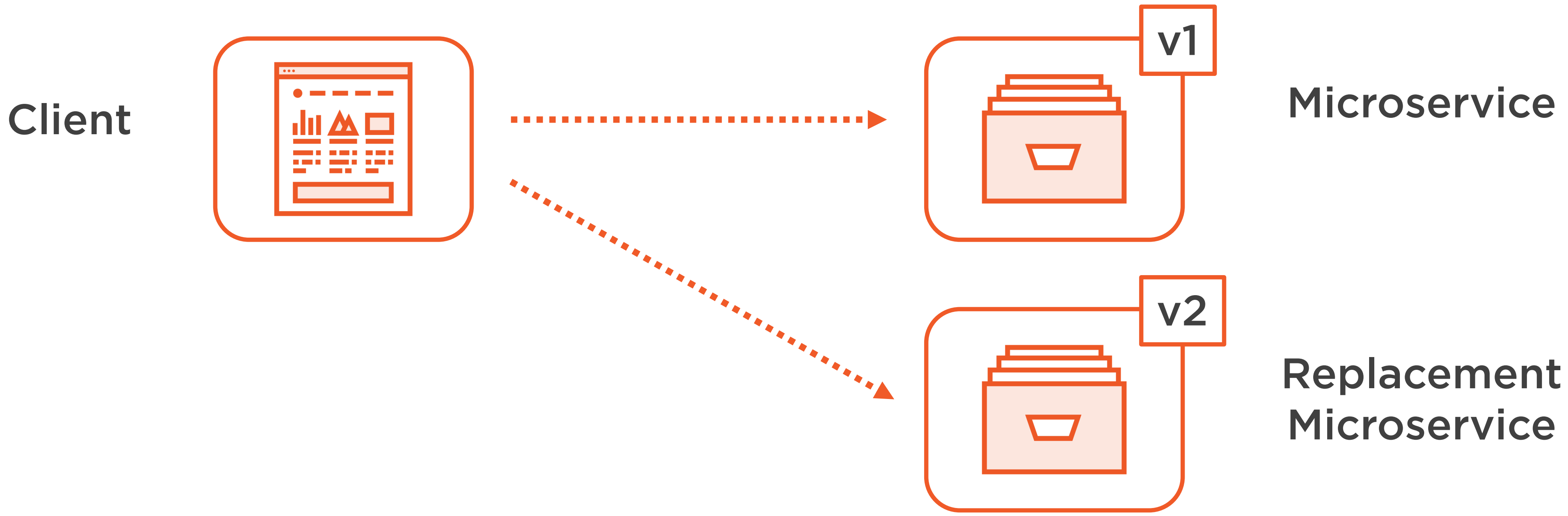


Implementing Advanced RESTful Concerns with ASP.NET Core 3 (Kevin Docx)

Replacing Microservices

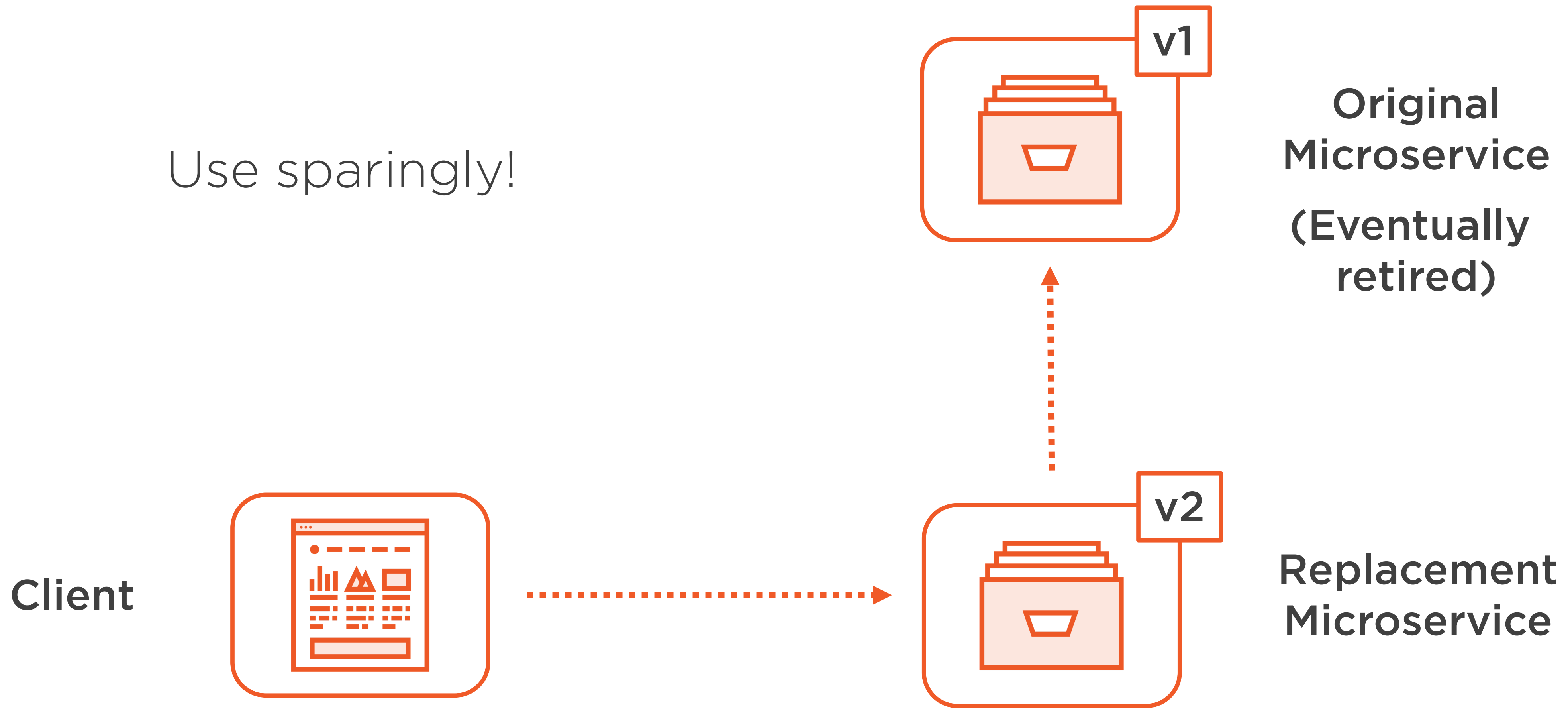
Create a brand new microservice to implement the new version of your API

Get away from “technical debt” or legacy technology



Incremental Migration

Use sparingly!



There is no “official” best way to implement versioning

ASP.NET Core Versioning

Sensible defaults out of the box

Logging

Config

Health Checks

Dependency Injection

NuGet package: Microsoft.AspNet.Core.Versioning

Supports versioning in path, query string or header

Supports optional version number

Summary

Maintaining backwards compatibility

Additive changes are safe

Other changes break clients

Don't assume you can force clients to upgrade on demand

Versioning strategies

Microsoft.AspNet.Core.Versioning NuGet package

Up next…

Implementing API Versioning