Implementing Security with API Gateway and BFF Patterns

Kevin Dockx
ARCHITECT

@KevinDockx https://www.kevindockx.com

Coming Up

Exploring the API gateway
- API gateway security pattern

Using Ocelot and integrating it with our identity service

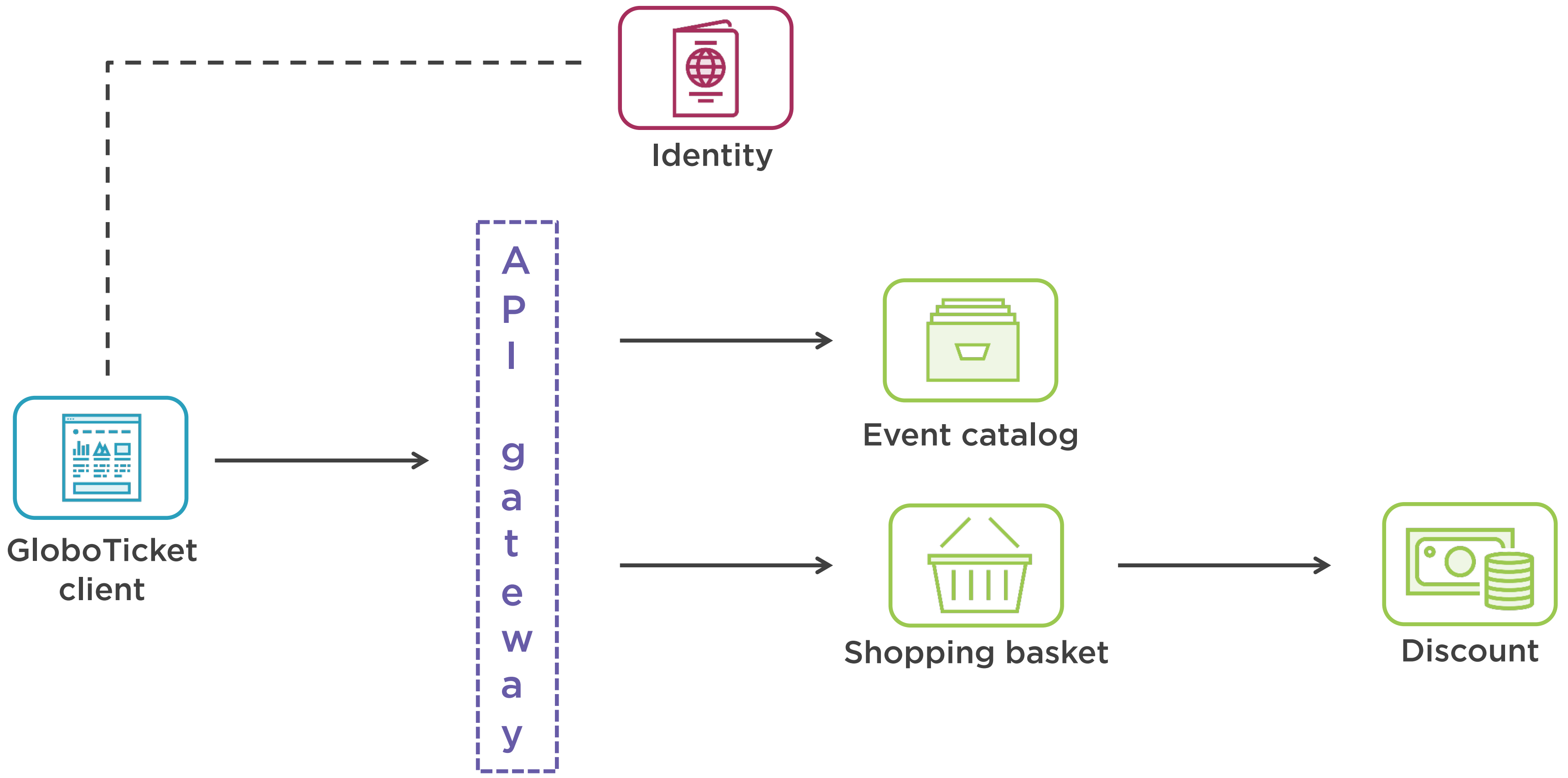
Passing user information downstream

The backend-for-frontend pattern

API gateway

An API management tool that sits between one or more client applications and one or more APIs

Exploring the API Gateway



Common API Gateway Tasks

Service discovery and aggregation

Rate limiting

Monitoring usage, analytics, logging

Caching

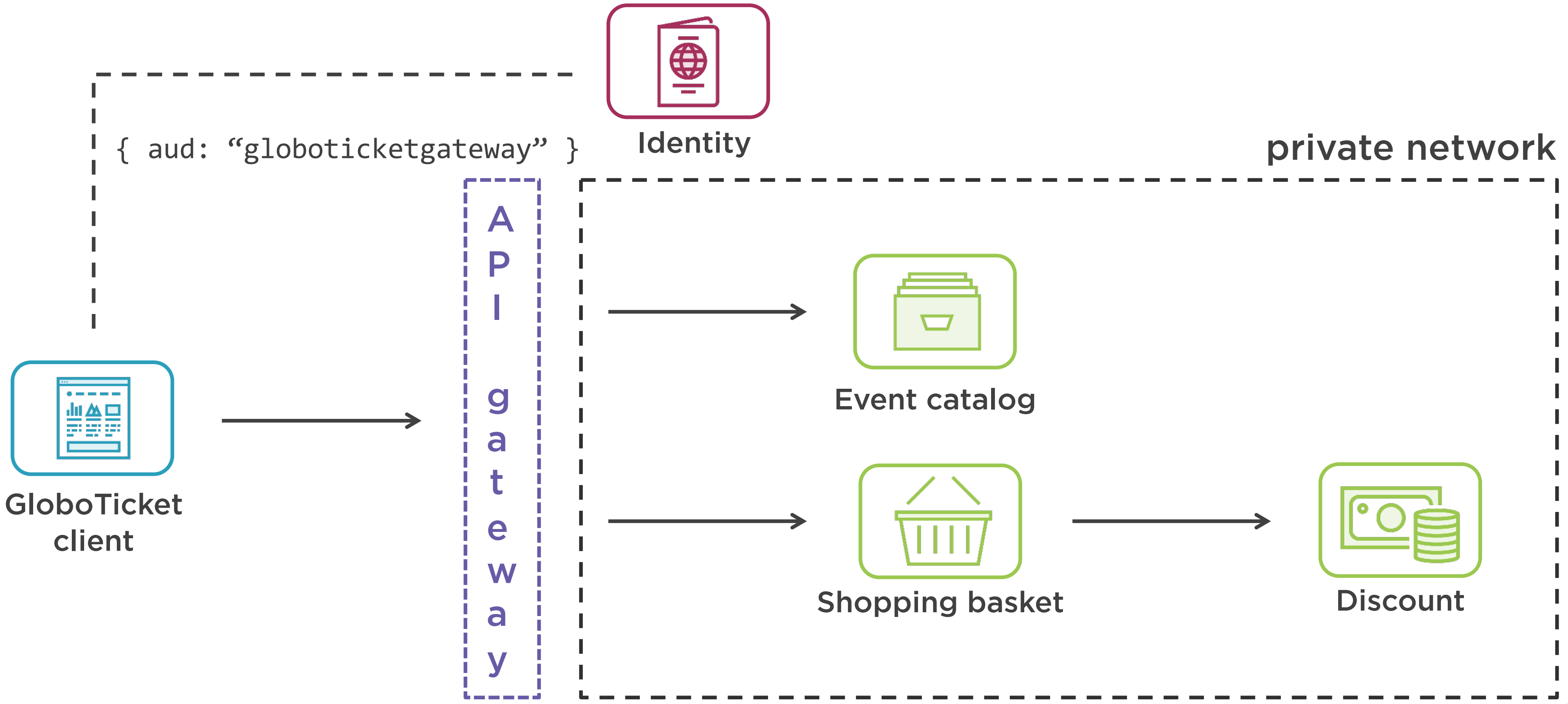
Handling security

Exploring the API Gateway

An API gateway

- Decouples the client from the backend implementation
- Takes away responsibilities

A Common API Gateway Security Pattern



HTTPS everywhere

HTTPS isn't just for the outside world. It's also for your internal, private network, cloud-based or otherwise.

Introducing Ocelot

A very simple ASP.NET Core project that passes through requests can be considered a bare-bones API gateway

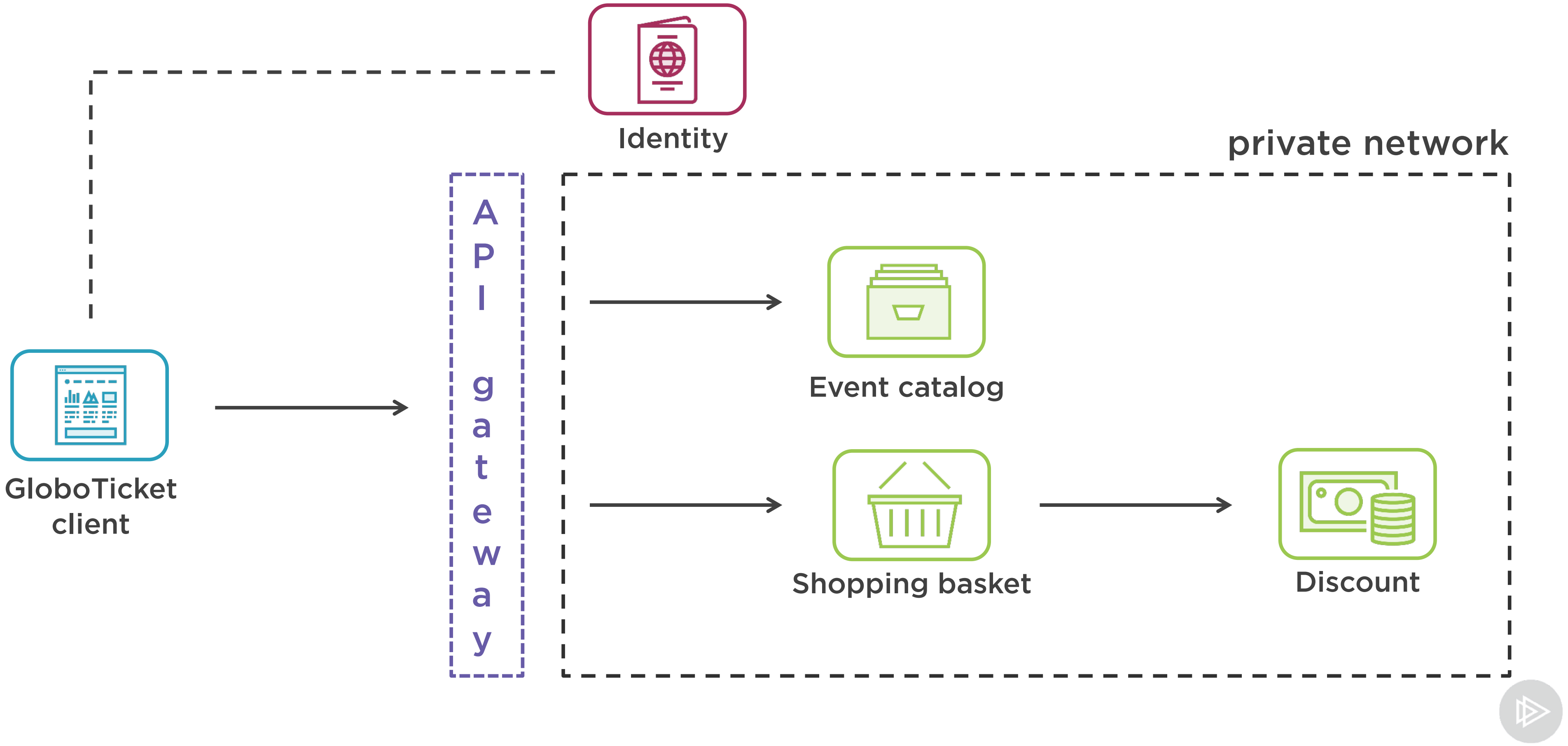
On Azure, Azure API Gateway is a very good option

- https://azure.microsoft.com/en-us/services/api-management

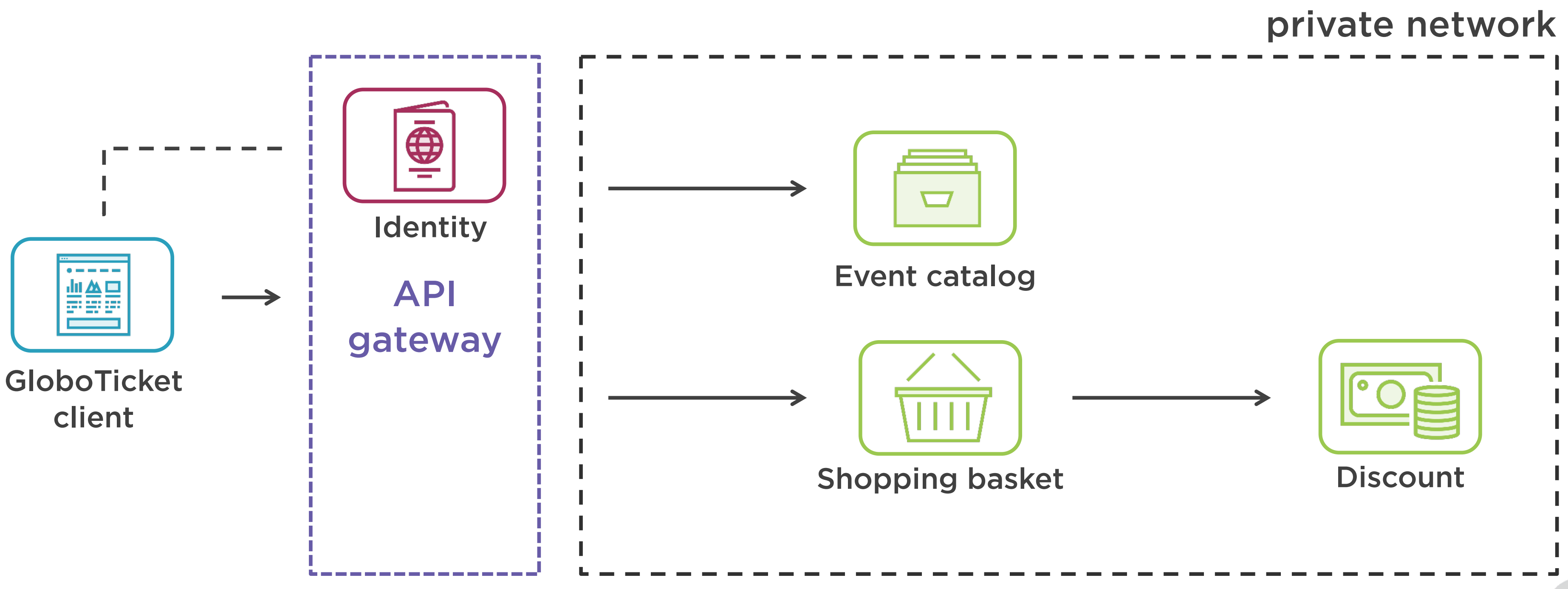
Ocelot is an open source .NET Core based API gateway

- Consists of a set of middleware that handles common tasks related to API gateways
- https://ocelot.readthedocs.io

Identity Service Location



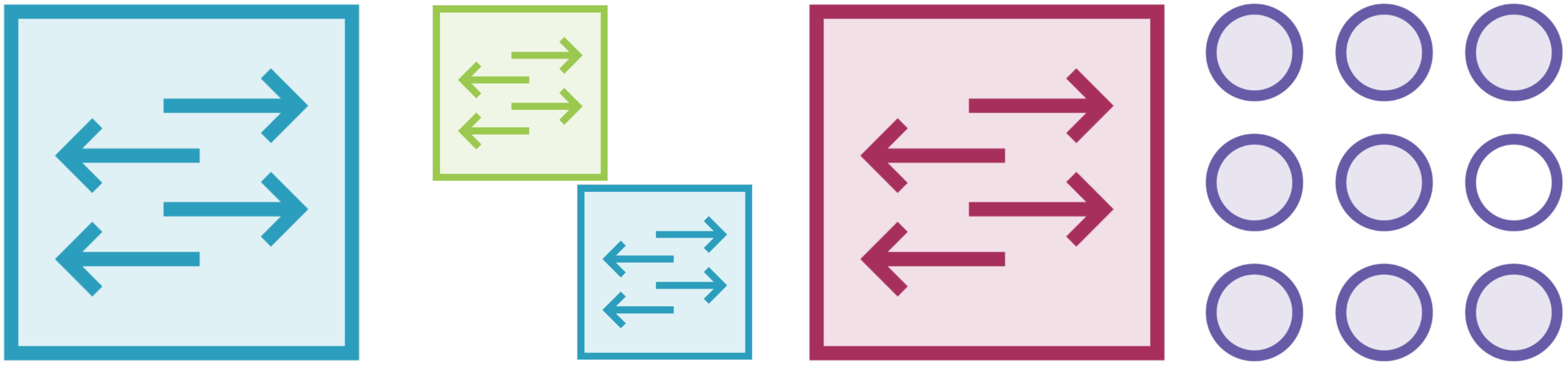
Identity Service Location



Identity Service Location

You identity service is used across your application landscape

Identity Service Location



One gateway across all APIs

Multiple gateways for multiple sets of APIs

Backend-for-frontend API gateways

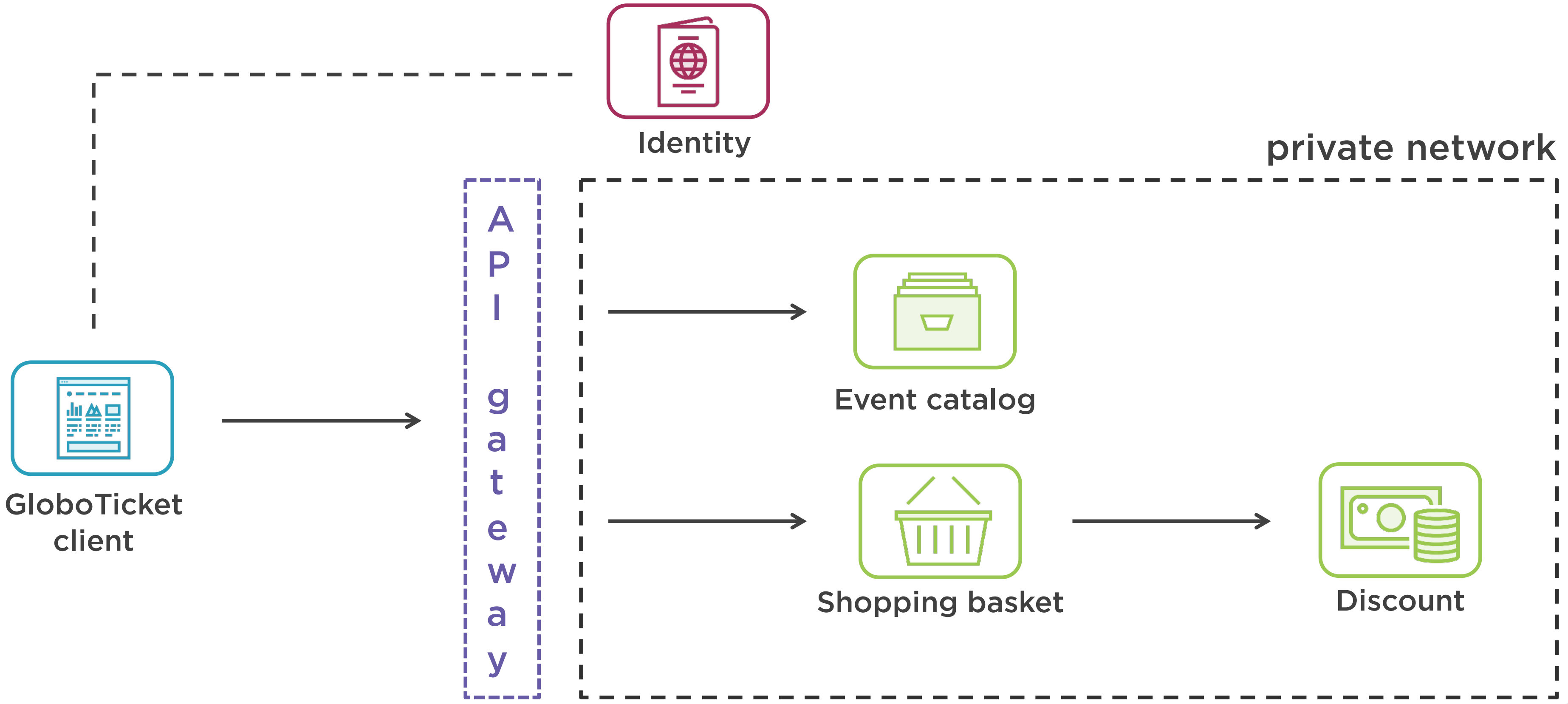
Other approaches

Identity Service Location

You identity service is used across your application landscape

- Don’t tie it to one API gateway

Introducing Ocelot



Demo

Adding Ocelot

Demo

Integrating Ocelot with our identity service

Passing User Information to a Microservice

Currently, the access token is not verified downstream as that's not the responsibility of the microservice in our approach

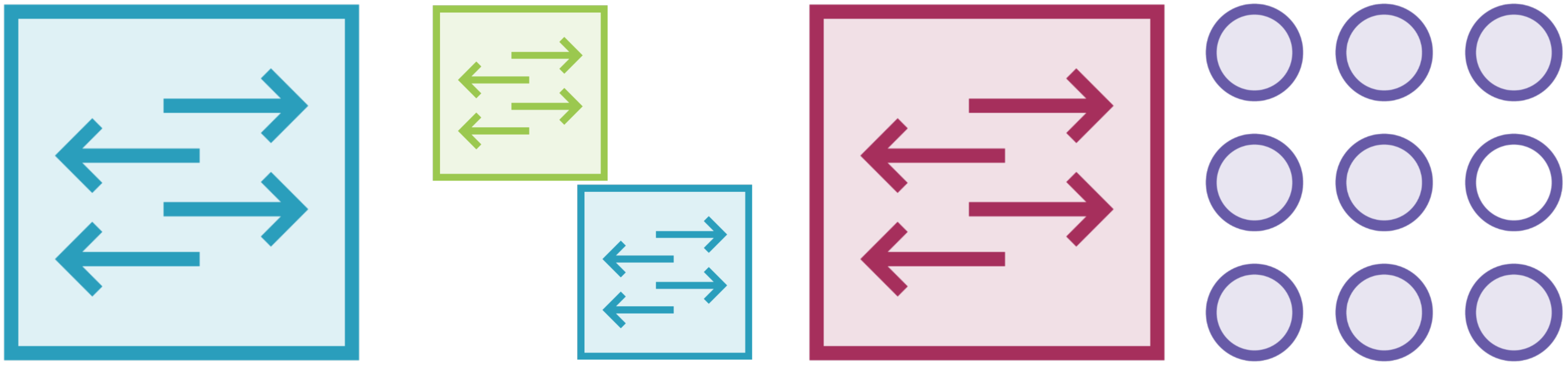
But…

- We can trust the user at level of the gateway
- We can pass that information downstream via request headers

Demo

Passing user information to a microservice

The Backend-for-frontend Pattern



One gateway across all APIs

Multiple gateways for multiple sets of APIs

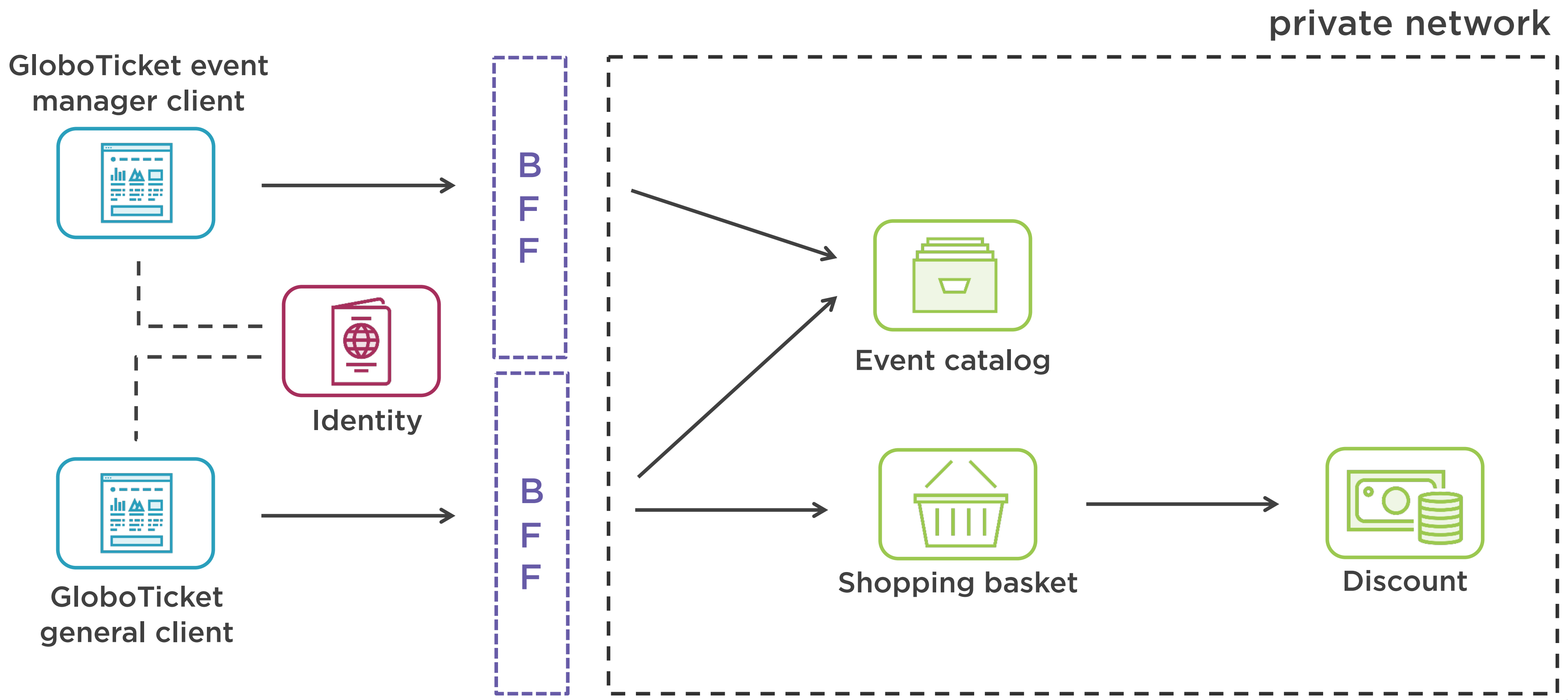
Backend-for-frontend API gateways

Other approaches

Backend-for-frontend (BFF)

A layer between the user experience and the resources it calls on, catered to each specific user experience

The Backend-for-frontend Pattern



The Backend-for-frontend Pattern

A backend-for-frontend is often an API gateway in its own right

- The BFF becomes the API gateway catered to a specific client or user experience

Summary

An API gateway is an API management tool that sits between one or more client applications and one or more APIs

- Service discovery and aggregation, monitoring, monetization, logging, rate limiting, … and security

Ocelot is an open source API gateway based on .NET Core

Summary

Gateway security pattern

- Check authentication and authorization at level of the gateway
- Microservices are not responsible for checking incoming tokens
- Everything behind the API gateway is secured on another level than application level

Summary

Ocelot specifics

- Secure routes to microservices by letting Ocelot check the incoming token
- Pass information to each microservice via request headers

A backend-for-frontend is a backend catered to a specific user experience

- It can be implemented as an API gateway