API Gateway made easy with Ocelot and Containers

Roberto Freato – *Consultant / Author / Solution Architect*

Building APIs

- «we already have an API, just expose it to the public»
  - *no, **protect it behind a revese proxy** to plug additional logic before clients reach it*
- «we are developing a new API, think about it as it would be the most complete and broader one»
  - *no, with the assumption **it would change frequently** in the (even short-term) future*
- «we split our platform into several micro-services/apis, but with common libraries and common design patterns»
  - *no, **each service will follow its own guidelines** to be completely independent*
- «in case we need a new API, we will upgrade the existing clients to use the new version»
  - *no, we will **create a new version keeping the client un-aware** of it*

Common Features

- Reverse proxy – Routing
- Logging, Monitoring and Analytics
- Errors management
- Documentation and API Portal
- Traffic Management
- Authentication
- UI Composition / Aggregation
- Transformation
- API Lifecycle / environments
- Caching and acceleration

API Gateways

- Commercial products:
  - Apigee (Google)
  - Layer 7 (CA Api Gateway)
  - WSO2
  - MuleSoft
  - AWS API Gateway
  - Azure API Management
  - Axway
- Open source alternatives

Ocelot

.NET Core API Gateway

https://github.com/ThreeMammals/Ocelot

Getting started

Scenario presentation and basic route-only implementation

Learned

- Ocelot delivered as NuGet package
- Needs:
  - «ocelot.json» configuration file
  - Basic declarations
- Basic route
- Redirection issues

Proxying APIs

Proxying Customers and Sales api with version-aware paths

Learned

- Versioning with url prefix (alternatives?)
- SSL termination is good
- Routes can be overlapping (with priorities)

ViewModel composition

Aggregate requests and project the client-optimized model

Learned

- Define dedicated paths for aggregation
- Use custom aggregation functions
- Minor fixes with headers transformation

Throttling

Traffic limiting under certain conditions

Learned

- Very basic throttling features
- Client-dependent feature (header)
- Relate to Authorization header (with limitations)

Extension

Extend Ocelot from outside (without pull requests)

Learned

- Simple DI-based configuration extension
- Makes the API Gateway stateless
- *Not so easy to extend/modify the pipeline*

Containerization

Azure App Service for Containers - PaaS

Staging/Production

Using the Gateway to discriminate environments

Learned

- Integrated with fully-managed Container Registry
- Shared, managed machine pool
- Single or multiple container support (with limitations)
- Integration of App Service with custom DNS
- Ocelot host-header-based routing

API Gateway made easy with Ocelot and Containers

Roberto Freato – *Consultant / Author / Solution Architect*