Architecture Report: API Gateways

Gleicon Moraes
https://github.com/gleicon
https://medium.com/@gleicon

Overview

API Gateways

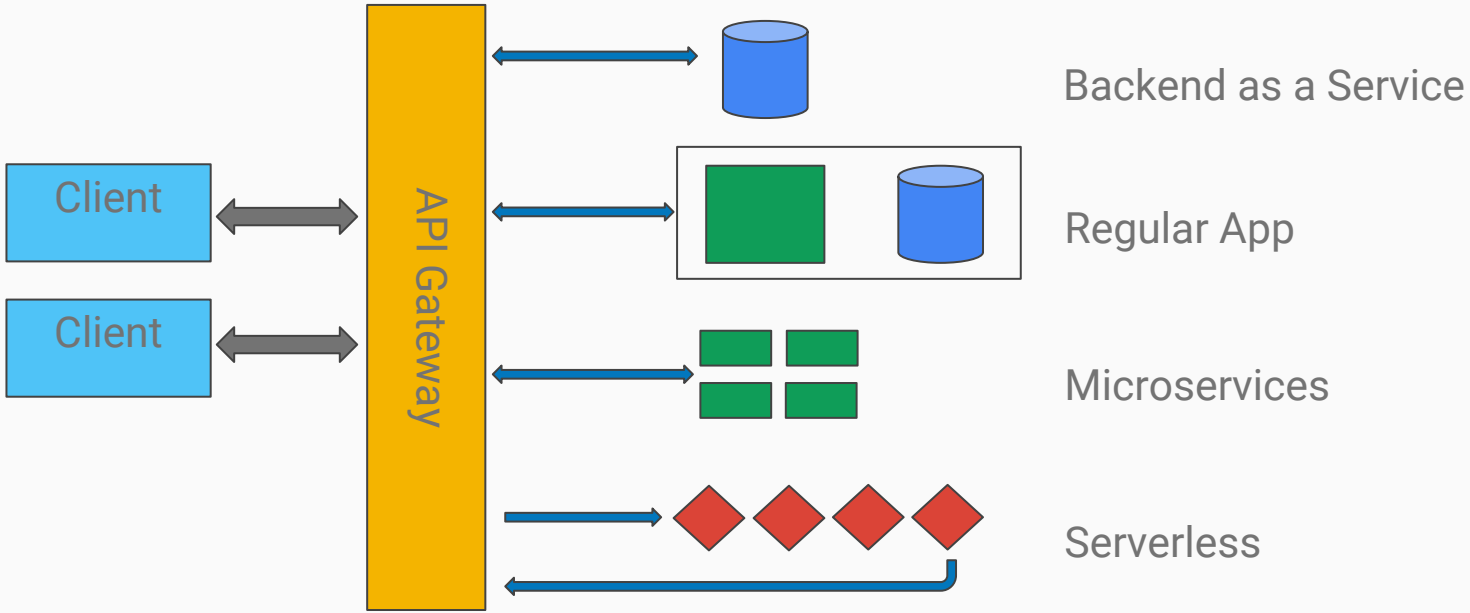
API Gateway is used to front and distribute access to internal APIs. Different vendors ship distinct building blocks and integrations. They can be SaaS or be installed in your infrastructure.

The API Gateway Pattern

API Gateway Pattern

- Separation of concerns between client and server code
- Distinct API views and responses from the same origin
- Call composition between APIs
- Single point of access
- Leverage migration from legacy code
- Leverage breaking monolith to microservices

Architecture diagram



Core features

Uniform authentication

- Allow for distinct frameworks and authentication models in the backend, acting as a translator
- Enable per user or per app credentials, granular control and logging
- Add security for legacy APIs

Rest over HTTPS

- SSL termination for legacy apps
- Single point of certificate maintenance
- Widespread transport protocol, compatible to most web frameworks

Horizontal scalability

- Composition of load balancing and application server
- No single request serving point, no sticky bit
- Scale up charged per request or per network traffic
- Distributed Request/Response caching

Payload rewrite

- Request/response intercept
- DSL for inline real time payload rewrite
- Mainly default to json handling

Request composition

- Combine two or more API responses into a single request
- Either concatenate or rewrite the response into a single payload
- Microservice calls
- Serverless event trigger

Backend as a Service

- Abstract database queries to API routes
- Manage connection pools and integration into legacy systems
- Batch call stored procedures
- Integration to message bus and messaging solutions
- Serverless architectures (e.g. AWS Lambda)

Analytics

- Detailed reports on requests
- Billing management
- Per request/origin error codes
- Volumetric analysis

Security

- DDOS Protection
- Throttling and rate limiting
- Whitelist and Blacklists
- Worldwide presence

Players

How to pick the right one

- Not all features are needed for your deployment
- Pick and choose based on budget and in house knowledge
- Migrate from an existing reverse proxy based on features
- Hosted or SaaS: evaluate the team workload and devops skill set
- ** Features may change - conduct your analysis based on features you need

Players

SaaS

- APIGEE
- AWS API Gateway
- Sensedia

Opensource

- Kong
- Tyk
- Nginx + Lua (openresty)

Comparison chart

	Uniform Authentication	Rest over HTTPS	Horizontal Scalability	Payload Rewrite	Request Composition	BaaS/ Serverless	Analytics	Security
APIGEE								
Amazon API GW								
Sensedia								
Kong								
Tyk								
Openresty								

Pitfalls

Dependency

BaaS and in house development

Until a contract renewal or in the face of a vendor change, dependency on BaaS might have created space so other development tasks were taken by the team. Take care to not outgrow your backlog counting on a long term contract. Alocate BaaS dependency as technical debt.

Authentication model

Engage into market proven authentication models that might be present in other players. Most APIs stick to OAUTH2 or AUTH Secrets.

Lock in

Pluggable infrastructure

IaaS vendors offer API Gateways that have competitive price models but make sure that you can serve and front APIs outside of their infrastructure for an attractive network ingress/egress price.

Development model

The configuration and coding should be versionable outside of the tool and automatable - meaning you should have tests for your configuration changes. The Gateway should support dev, integration and production environments and version promotion.

Scalability

Horizontal scalability

SaaS model: use the bundled analytics and instrument your backend. If you host your
Hosted model on cloud: make sure you reserve the minimum or regular day to day usage and pick marketplace prices to lower the impact on your monthly rates
Hosted on VPS/bare metal: plan for at least 30% peaks on each server.

Backend savings

API gateway should save backend computing and network resources. Make sure you review your sizing and scaling. Do use automation to ensure resources grow according to the workload.

Conclusion

1. Use API Gateways as accelerators but plan for reducing their footprint on your architecture
2. Use standard transport and authentication protocols
3. Take care with cache and data coherency
4. Strive to be able to migrate to at least one open source solution