


00-API_Gateway

Introduction

This note gather information obtained during my discovery of **What is a API Gateway?**.

Security attention points

All element marked with  are points that must taken in account from a security point of view.

Misc

- Table were generated with [this tool](#).
- HTTP client used was [this one](#).

Study roadmap

- ✓ Implement the labs i.e. all the API definitions.
- ✓ Create the security tests case to validate all the API configuration using VENOM test plans (one test plan by type of API).
- ✓ Use the APIMAN admin API to auto provision the labs and its configuration from scratch: Create a docker compose file to link the both container APIMAN + Demo sandbox
- ✓ Create a XLM blog post about all this study.

Data source

- [PDF](#) file was used for the study.
- Images were taken from [the crash course html content](#).

Note for the blog post

The blog post will have the following section:

1. Explosion of API usage.
2. What is a API Gtw and its objective?
3. Use a API Gtw is good but how to ensure that api config is always secure?
4. POC and approach explanation
5. Conclusion

Lab

Details

See [here](#).

Github repository

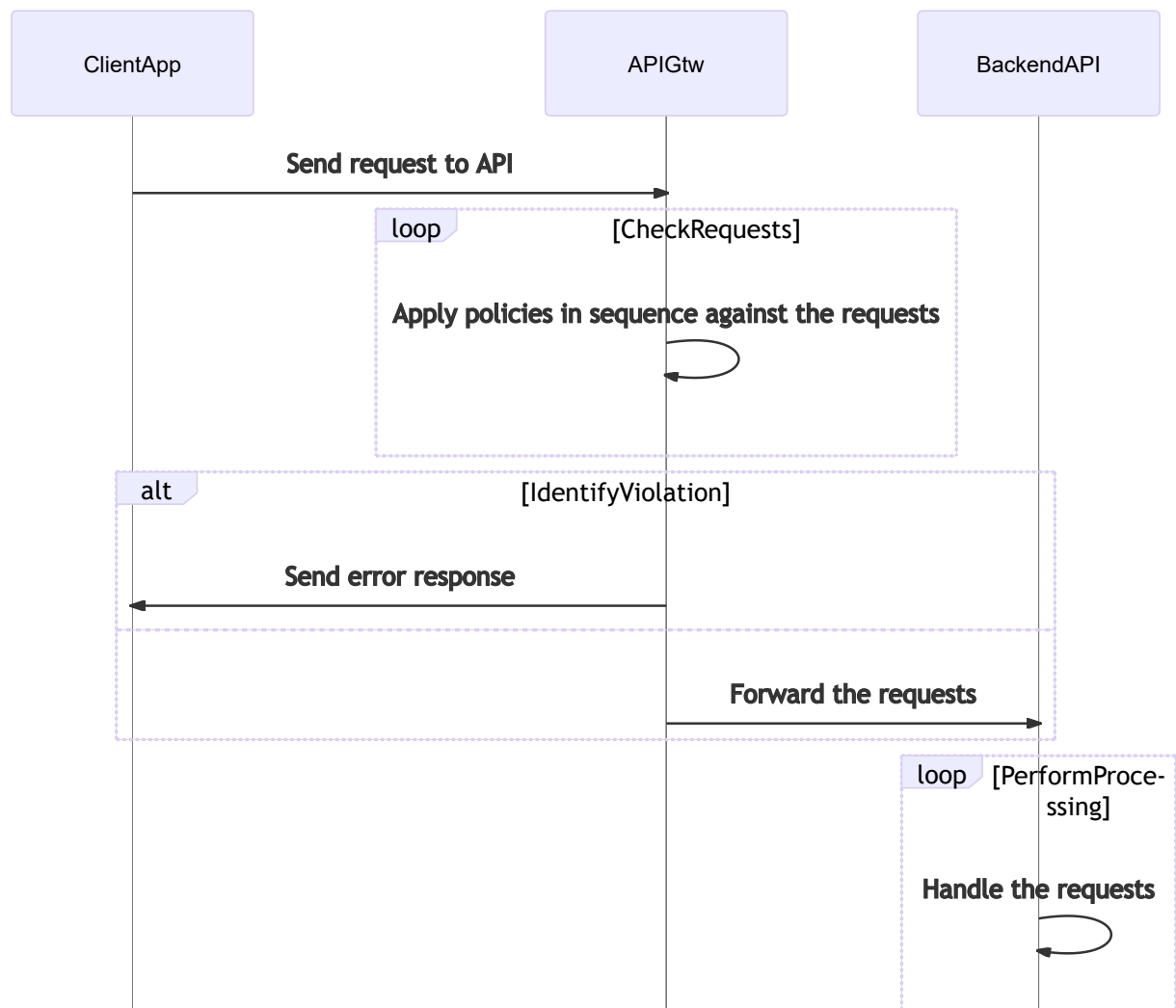
See [here](#).

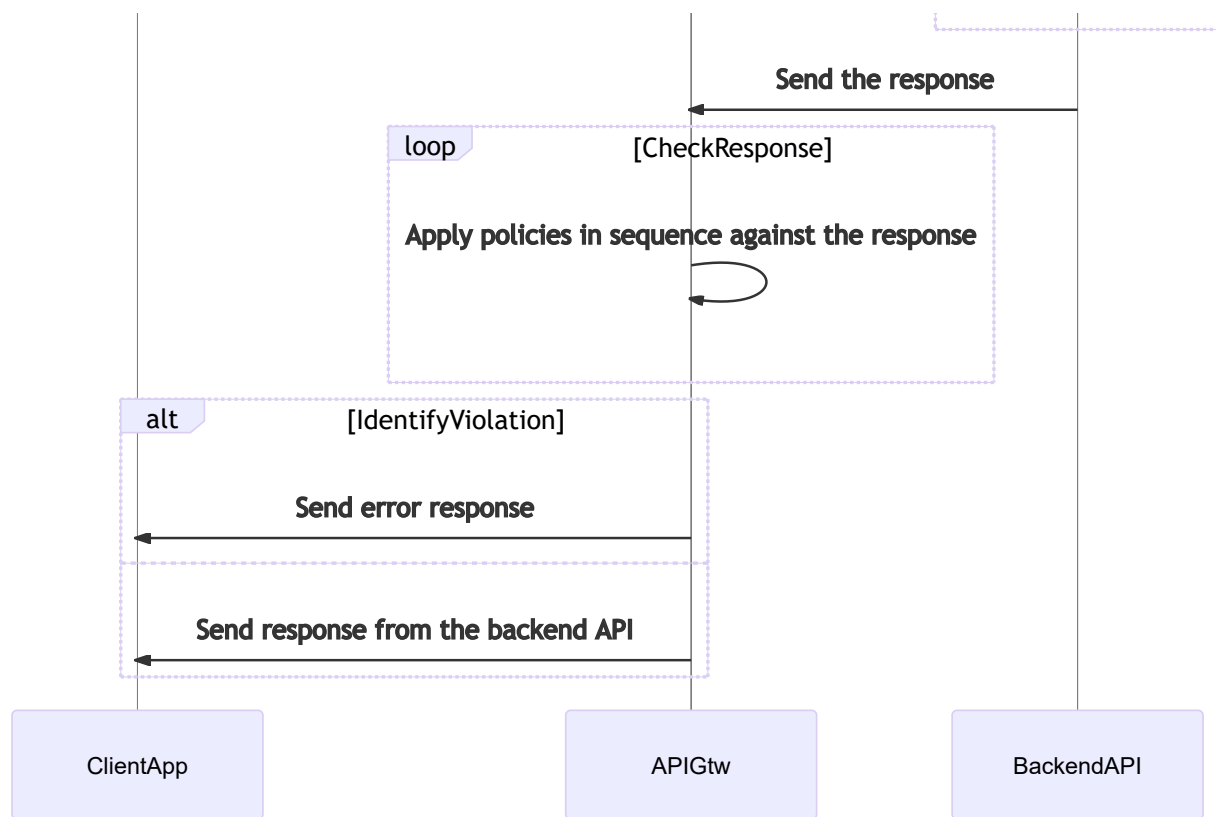
Study notes

API Gtw role

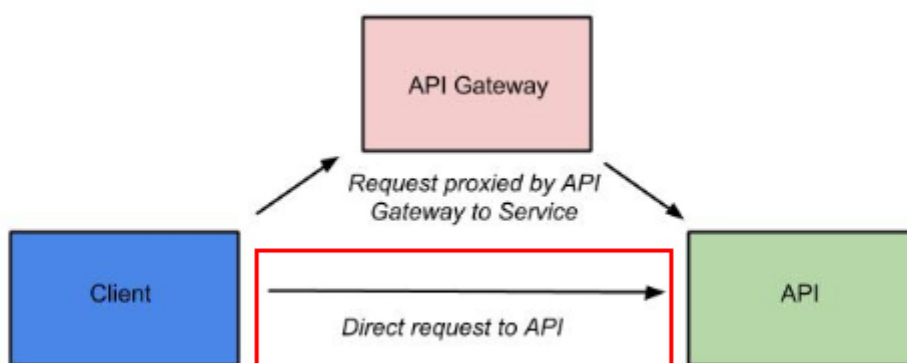
The objective of the API Gtw is to be a central access point for any call to an API offered to client apps so to be effective all call to API must be routed (at network level) to the APi Gtw. In this way, Backend API can delegete several tasks to the API Gtw like authentication, authorization, rate limiting, quota, etc.

Communication flow:





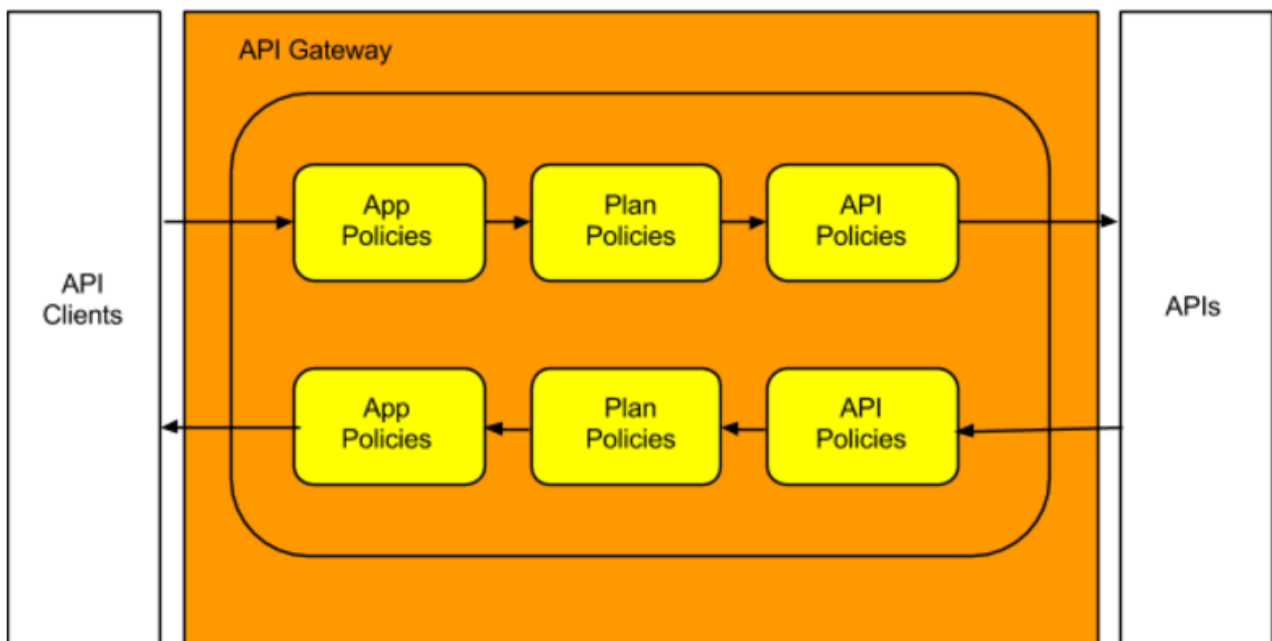
⚠ The main drawback is that in case of configuration issue on the network level, for example, if backend API can be called correct, then it's game over (in red):



Policies

- A policy is a set of rules to apply on a request or a response.
- Policies are applied for the incoming request from the client app and the response received from the backend API:

API Gateway - The 2-Way Policy Chain



Publishing API

👁 Important point regarding *PUBLIC API* vs *PLANS API* in APIMAN:

PUBLIC API:

"Public APIs are also very flexible in that they can be updated without being re-published. Unlike APIs published through Plans, Public APS can be accessed by a client app without requiring API consumers to agree to any terms and conditions related to a contract defined in a plan for the API. It is also important to note that when an API is Public, only the policies configured on the API itself will be applied by the API Gateway."

PLANS API:

"Publishing an API through Plans - In contrast to Public APIs, these APIs, once published, must be accessed by a Client App via its API key. In order to gain access to an API, the Client App must create a contract with an API through one of the API's configured Plans. Also unlike Public APIs, APIs that are published and accessed through its Plans, once published, cannot be changed. To make changes, new versions of these APIs must be created."

Demo API definition

APIMAN entire configuration export is [here](#).

Organization for all API is `XLM`.

Published API - Blog API

- Status: **Ready to be used**.
- Plan: `standard`.

- Backend API: `http://jsonplaceholder.typicode.com/`.

- Policies:

Policy name	Level	Goal
IP Whitelist	Plan	Only allow requests from <code>127.*.*.*</code> and <code>172.*.*.*</code> .
Rate Limiting	Plan	Only allow 10 requests by minute by client app.
Basic Autentication	API	Require basic authentication from users defined statically (creds: <code>user/password</code>) and forward the login to the backend API in header <code>X-User</code> .

Call syntax:

```
# Call raising an missing authentication error
$ http --verify=no "https://localhost:8443/apiman-gateway/XLM/blog/1.2/todos/1?apikey=d09e70b2-2abc-47d8-9168-80878e662e6a"
...
# Successful call
$ http --verify=no -a user:password "https://localhost:8443/apiman-gateway/XLM/blog/1.2/todos/1?apikey=d09e70b2-2abc-47d8-9168-80878e662e6a"
...
```

Public API - Bin API

- Status: **Ready to be used.**
- Plan: None because it is public.
- Backend API: `https://requestbin.net/r/`.
- Policies:

Policy name	Level	Goal
CORS	API	Only allow origins <code>https://localhost:8443</code> , <code>http://localhost:8080</code> and apply cache of 10 seconds.
HTTP Security	API	Enable CSP / X-Frame-Options / X-Content-Type-Options headers. Disable HSTS (local POC) / X-XSS-Protection headers.
Simple Header	API	Remove following headers from backend API response: cf-request-id, Report-To, Server, CF-RAY, NEL, CF-Cache-Status.

Call syntax - replace `[BIN_ID]` by the <https://requestbin.net> BIN identifier:

```
# Call raising a CORS origin not allowed error
$ http --verify=no "https://localhost:8443/apiman-
gateway/XLM/bin/1.0/[BIN_ID]?a=b" Origin:https://localhost:8442
...
# Valid call
$ http --verify=no "https://localhost:8443/apiman-
gateway/XLM/bin/1.0/[BIN_ID]?a=b" Origin:https://localhost:8443
...
```

Venom test plans

Each test plan will ensure that the API configuration do the expected job.

Tools used:

- [JQ](#).
- [VENOM](#).

💡 One test dedicated test plan by API was created to made maintenance and evolution more easier.

Venom do not resolve variable between them in the `vars` block so it's the reason why i was force to use variable for the API Gtw host and put explicitly URL in all test cases. This to allow me pass the API Gtw host as external variable.

Run:

```
PS> venom run --format="json" --output-dir="." [TEST_PLAN_FILENAME].yaml
| Out-Null
PS> Get-Content .\test_results.json | jq --raw-output --sort-keys
'.test_suites[].testcases[] | select(.failures != null).name'
```

Published API - Blog API

Venom test plan file named [published-api-test-plan.yaml](#).

Public API - Bin API

Venom test plan file named [public-api-test-plan.yaml](#).