**Proxy Chaining in Apigee**

**Proxy Chaining** in Apigee refers to the process of having **one API proxy** call **another API proxy internally**, rather than directly calling an external backend service.

**How It Works**

* **Main Proxy (Proxy-A):** This is the public-facing API that clients call.
* **Internal Proxy (Proxy-B):** This proxy handles reusable logic (like security checks, formatting, logging, etc.).

Instead of Proxy-A calling an external URL, it uses:

<LocalTargetConnection>

<APIProxy>proxy-b</APIProxy>

<ProxyEndpoint>default</ProxyEndpoint>

</LocalTargetConnection>

This means:

* **Proxy-A sends the request to Proxy-B internally** (within Apigee)
* **Proxy-B either returns a response or calls an external backend**

**Use Case: Centralized Security Proxy for Shared Authentication**

**Scenario**

You manage **multiple internal APIs** that perform different business functions, such as:

* orders-api – handles order creation and tracking
* inventory-api – manages product availability
* user-api – deals with user data

All of these APIs need to:

* Authenticate the incoming request (OAuth 2.0 or API Key)
* Perform rate limiting
* Do common logging

**Problem**

* Each API repeats the same logic for authentication, logging, and throttling.
* Maintaining that logic in **every proxy** increases **code duplication** and **operational overhead**.

**Solution: Use Proxy Chaining**

You can create a **central security proxy** (security-proxy) that:

* Validates the token/API key
* Applies rate limiting
* Logs request metadata

Then create a **main proxy** (orders-api, inventory-api, etc.) that **forwards requests** to security-proxy using <LocalTargetConnection> before continuing to the actual backend.

**Summary**

| **Aspect** | **Without Proxy Chaining** | **With Proxy Chaining** |
| --- | --- | --- |
| Duplication | High | Low |
| Scalability | Hard to maintain | Easy to scale |
| Security Control | Fragmented | Centralized and consistent |
| Performance | Redundant processing | Streamlined |

**RouteRule (Dynamic Backend Routing)**

**Scenario: Query Parameter-Based Routing**

You have a /weather API proxy that:

* For ?source=openweather, fetches data from OpenWeatherMap API
* For ?source=weatherapi, fetches data from WeatherAPI
* For no parameter, routes to a generic default API

**Why Use RouteRule Here:**

You want to **dynamically select a backend TargetEndpoint** based on the query parameter.

**Configuration:**

<RouteRule name="ToOpenWeather">

<Condition>(request.queryparam.source = "openweather")</Condition>

<TargetEndpoint>openweather-target</TargetEndpoint>

</RouteRule>

<RouteRule name="ToWeatherAPI">

<Condition>(request.queryparam.source = "weatherapi")</Condition>

<TargetEndpoint>weatherapi-target</TargetEndpoint>

</RouteRule>

<RouteRule name="Default">

<TargetEndpoint>default</TargetEndpoint>

</RouteRule>

**What it solves:**

* Routes requests to **different backends** based on query parameter
* No need to modify request/response in detail
* Keeps API proxy flexible for multiple backend integrations