**Flows and RouteFlow in Apigee**

**<Flows>**

* <Flows> is a container that holds one or more <Flow> elements.
* Each <Flow> represents a **conditional logic block** that applies to **incoming requests or outgoing responses** if the specified condition is met.

**Structure Explanation**

Here’s the snippet again:

<Flows>

<Flow name="v1-validation">

<Condition>proxy.pathsuffix MatchesPath "/v1/resource"</Condition>

<Request>

<Step>

<Name>ValidateQueryParams</Name>

</Step>

</Request>

</Flow>

<Flow name="v2-validation">

<Condition>proxy.pathsuffix MatchesPath "/v2/resource"</Condition>

<Request>

<Step>

<Name>ValidateJSONPayload</Name>

</Step>

</Request>

</Flow>

</Flows>

**Explanation of Each Part**

**1. <Flow name="...">**

Defines a named flow with a **specific condition**.

**2. <Condition>...**

This defines the **triggering condition**. The logic here means:

* If the **path suffix** (i.e., the part after the base path) matches /v1/resource or /v2/resource, then the corresponding flow will execute.

E.g., if the request is:

GET /api/v1/resource

Then proxy.pathsuffix = /v1/resource, so v1-validation flow will trigger.

**3. <Request> and <Response>**

These blocks define what to do **during the request or response phase** of the flow.

You can define:

* What policies to apply to the request
* What actions to take on the response

In this example, only the **request** section is being used.

**4. <Step> and <Name>**

A <Step> is an individual action within the flow.  
Each <Step> calls a **named policy** defined elsewhere in the proxy configuration.

**For example:**

<Step>

<Name>ValidateQueryParams</Name>

</Step>

This means:

* There is a policy file named ValidateQueryParams.xml inside the policies/ directory.
* Apigee will execute that policy when this flow runs.

**What would ValidateQueryParams.xml contain?**

Typically, something like:

<ValidateParameters name="ValidateQueryParams">

<DisplayName>Validate Query Parameters</DisplayName>

<Properties/>

<Parameters>

<QueryParam name="id">

<Required>true</Required>

</QueryParam>

<QueryParam name="type">

<Required>true</Required>

</QueryParam>

</Parameters>

</ValidateParameters>

**Summary of What This Block Does**

| **Flow Name** | **Triggered When Path is...** | **Applies Which Policy** |
| --- | --- | --- |
| v1-validation | /v1/resource | ValidateQueryParams |
| v2-validation | /v2/resource | ValidateJSONPayload |

**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