**Core Policy Implementation in Apigee**

**Core Policy Implementation in Apigee**

In Apigee, policies are reusable units of logic that modify the behavior of API proxies. Policies allow developers to implement common tasks like transformation, security enforcement, error handling, and flow control without altering the backend services. These are configured declaratively in XML and attached to specific flows in the API proxy.

Core policies simplify proxy behavior and promote a clean separation between business logic and service orchestration.

**1. Message Transformation (AssignMessage, JSON-to-XML, XML-to-JSON)**

**AssignMessage Policy**

The AssignMessage policy is used to create or modify request and response messages. It allows setting headers, payloads, HTTP verbs, and even target endpoints.

**Use Cases:**

* Set custom headers
* Rewrite URLs
* Modify HTTP methods (e.g., convert POST to GET)

**Example:**

<AssignMessage name="SetHeaders">

<Set>

<Headers>

<Header name="x-api-version">v1</Header>

</Headers>

<Verb>GET</Verb>

</Set>

<IgnoreUnresolvedVariables>true</IgnoreUnresolvedVariables>

<AssignTo createNew="false" transport="http" type="request"/>

</AssignMessage>

**JSON to XML Conversion**

Apigee offers built-in policies for format transformation between JSON and XML.

**Use Case:**

When a backend service accepts only XML, but the client sends JSON.

**Example:**

<JSONToXML name="json-to-xml-transformer">

<InputVariable>request</InputVariable>

<OutputVariable>xmlRequest</OutputVariable>

</JSONToXML>

**XML to JSON Conversion**

<XMLToJSON name="xml-to-json-transformer">

<InputVariable>response</InputVariable>

<OutputVariable>jsonResponse</OutputVariable>

</XMLToJSON>

**2. Error Handling and Fault Rules**

Error handling ensures that when something goes wrong during proxy execution (e.g., timeouts, transformation errors, failed authentication), the client receives a clean, consistent error message.

**Fault Rules**

Fault rules define how Apigee should respond when a policy or flow fails.

**Use Case:**

* Catch transformation or target errors
* Return standardized error messages

**Example: Global Fault Rule**

<FaultRules>

<FaultRule name="HandleErrors">

<Step>

<Name>RaiseFault</Name>

</Step>

<Condition>(fault.name Matches "JSONToXMLFailed")</Condition>

</FaultRule>

</FaultRules>

**RaiseFault Policy Example**

<RaiseFault name="InvalidPayloadFault">

<FaultResponse>

<Set>

<Headers>

<Header name="Content-Type">application/json</Header>

</Headers>

<Payload>{"error":"Invalid input format"}</Payload>

<StatusCode>400</StatusCode>

<ReasonPhrase>Bad Request</ReasonPhrase>

</Set>

</FaultResponse>

</RaiseFault>

**3. Flow Control with Conditional Flows**

Flow control is essential for managing the direction and logic of execution in an API proxy. Apigee enables conditional branching using <Condition> elements to execute different policies based on runtime context.

**Types of Flows:**

* PreFlow (before specific conditions)
* Conditional Flows (based on request paths or headers)
* PostFlow (always executed after conditional flows)

**Example: Conditional Path Routing**

<Flow name="product-flow">

<Condition>proxy.pathsuffix MatchesPath "/products"</Condition>

<Request>

<Step>

<Name>ProductRequestTransform</Name>

</Step>

</Request>

</Flow>

**Use Case:**

Different policies can be applied based on the URL path, such as:

* /login → apply OAuth token validation
* /orders → apply rate limiting
* /payments → apply JSON-to-XML transformation

This enables highly flexible and efficient API proxy design.

**Real-World Use Case: Travel Booking API**

A travel company offers APIs for booking, payment, and customer data. Apigee proxy implements:

* **AssignMessage** to rewrite URLs and set headers
* **JSONToXML** to convert modern mobile requests to legacy backend formats
* **FaultRules** to send standard 500 errors with user-friendly messages
* **Conditional Flows** to apply separate security logic for /admin routes

This configuration improves performance, enhances security, and ensures consistent client experience without touching backend services.

**Conclusion**

Core policy implementation in Apigee empowers developers to manage API behavior with minimal code. Message transformation ensures format compatibility, error handling promotes robust interaction, and conditional flows provide logical control over request/response paths. Together, these policies form the foundation of secure, scalable, and maintainable API proxy design.