I APIS

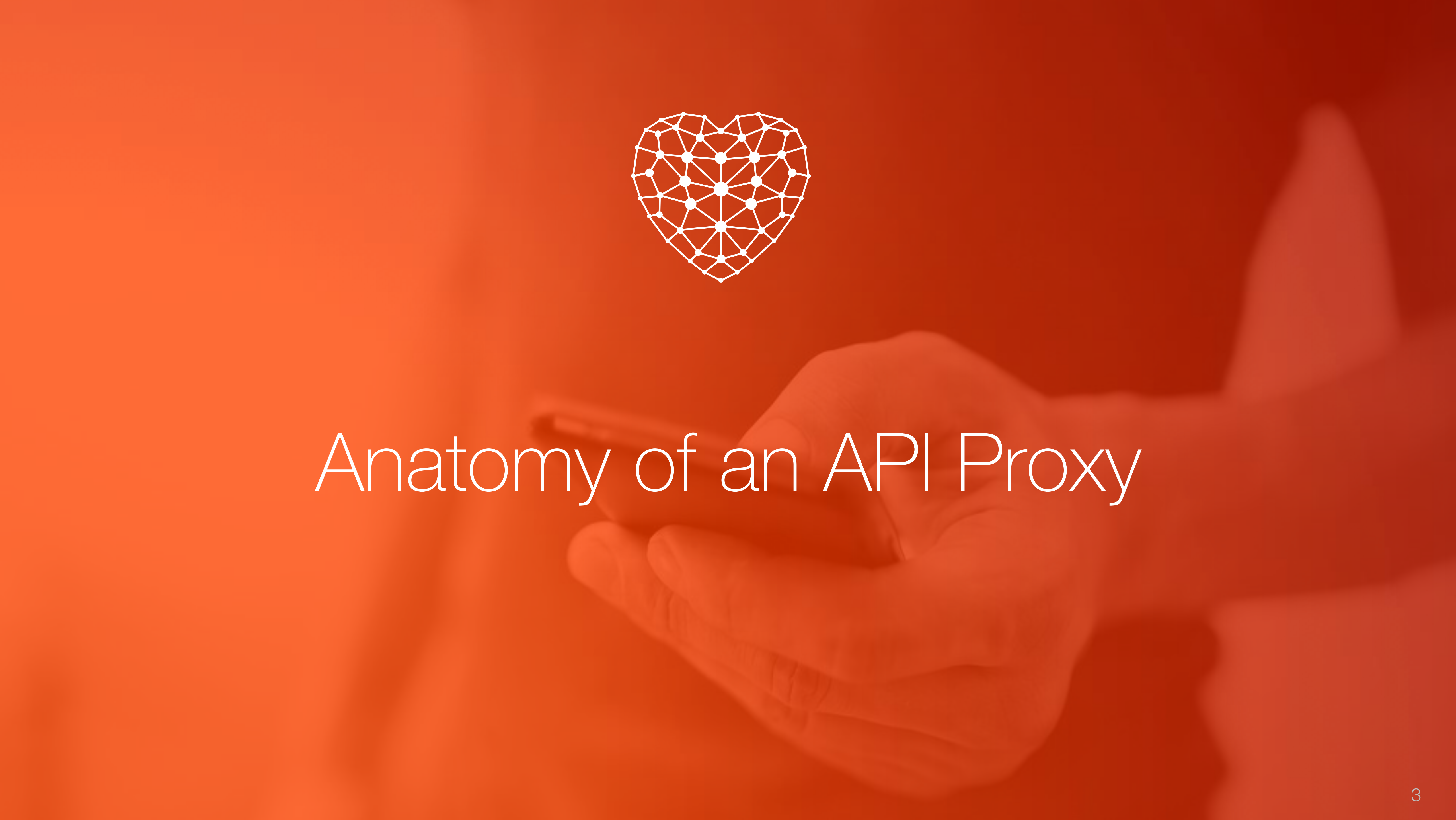
Crash Course: Foundational Topics in Apigee Edge

Steve Richardson
API Lumberjack

apigee

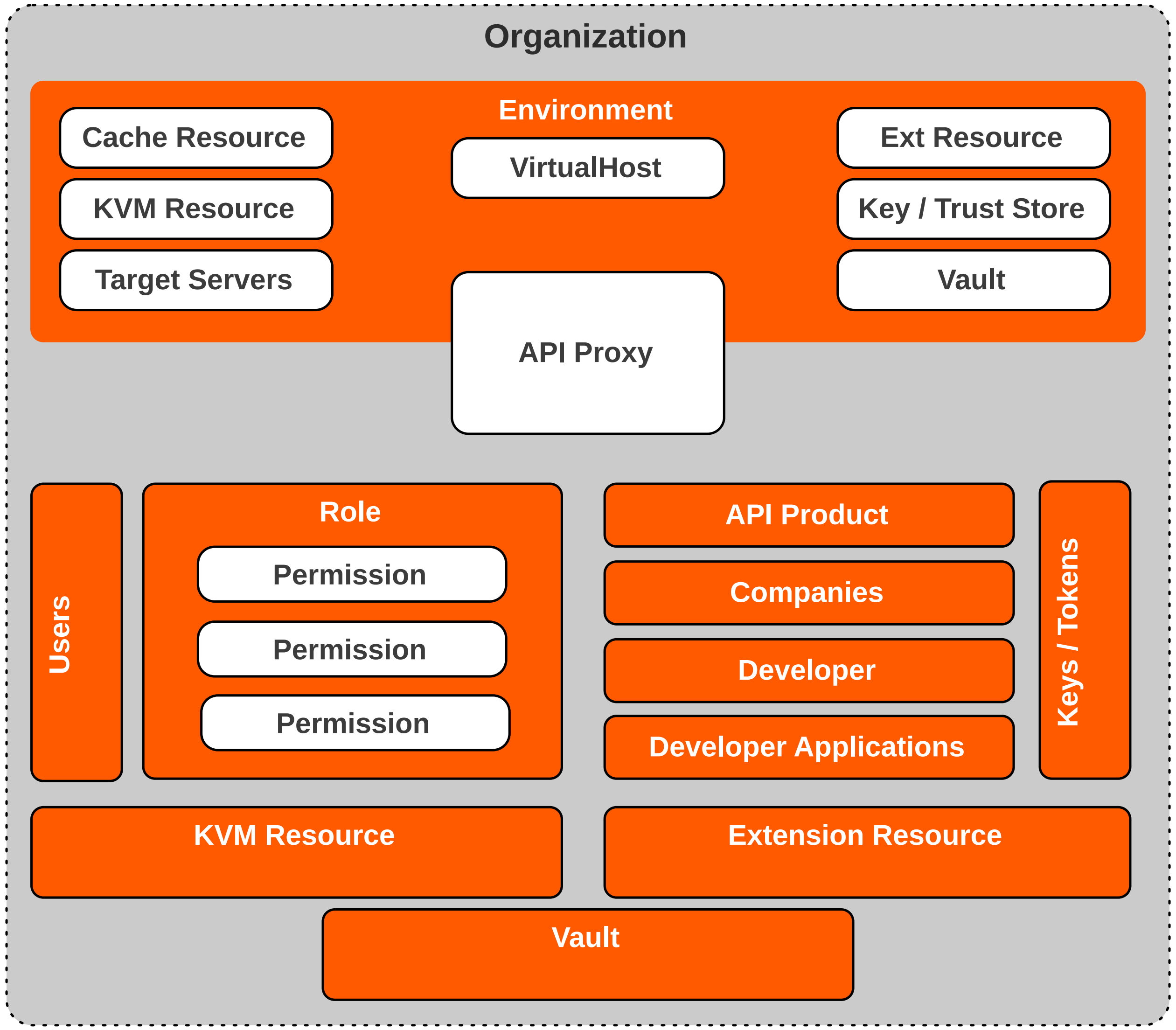
Agenda

1.	**Anatomy of an API Proxy**	**9:30**
2.	Connectivity	10:30
3.	API Management	12:00
4.	Rate Limiting	2:30
5.	Caching	3:30
6.	Custom Analytics	5:00

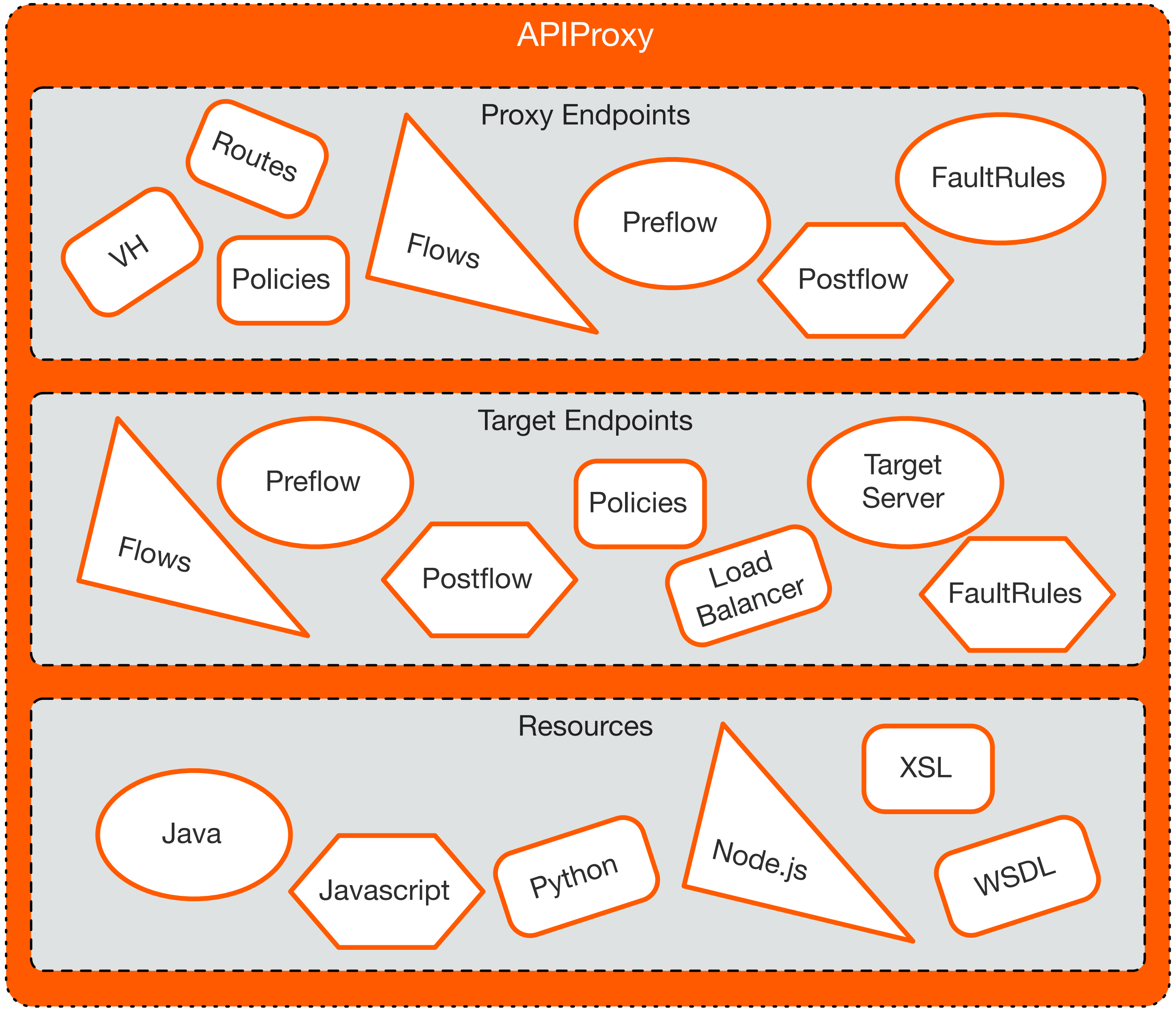


Anatomy of an API Proxy

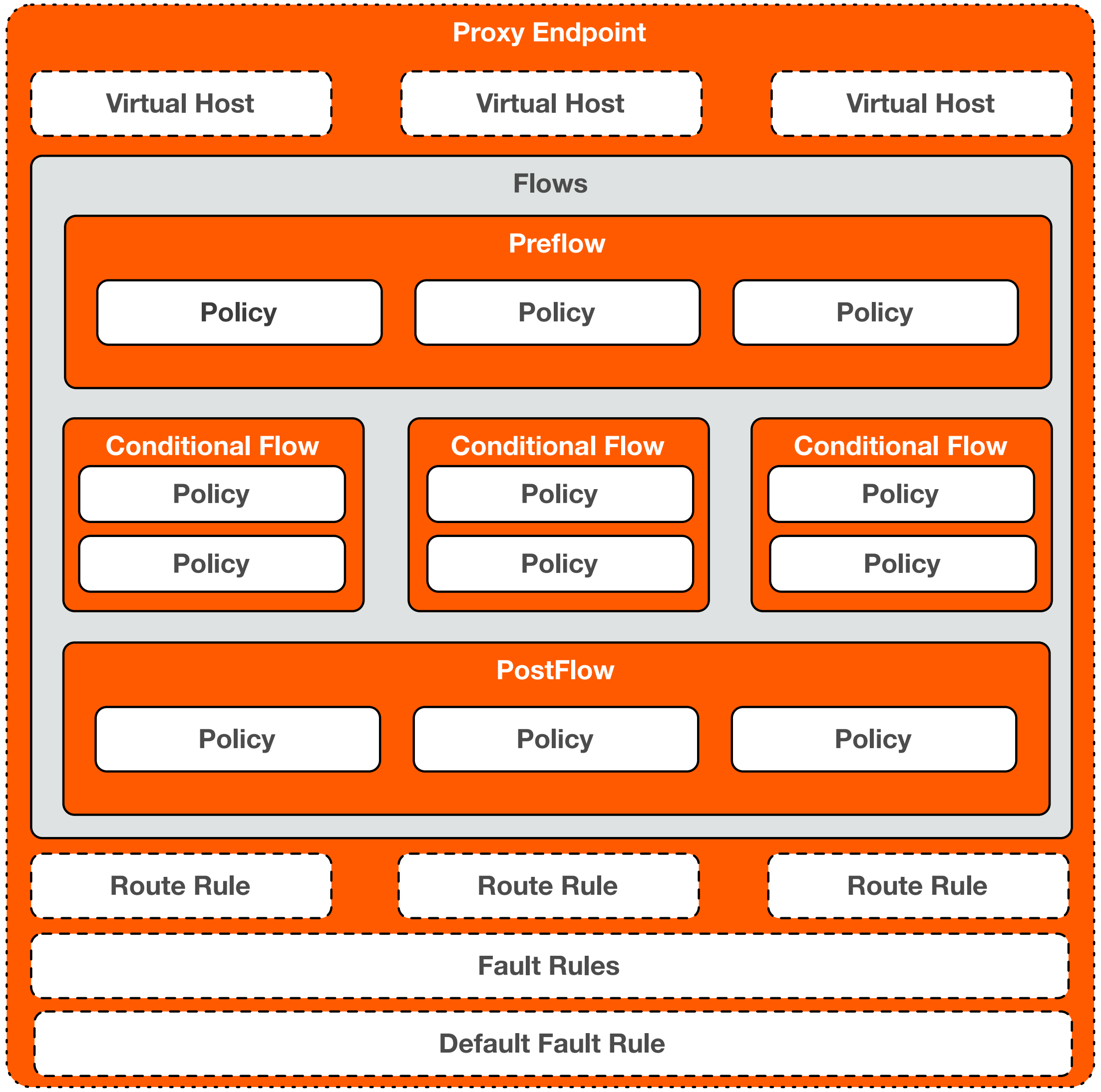
Platform Entities



API Proxies


APIProxy
Proxy Endpoints
Routes
VH
Policies
Flows
Preflow
Postflow
FaultRules
Target Endpoints
Flows
Preflow
Postflow
Policies
Load Balancer
Target Server
FaultRules
Resources
Java
Javascript
Python
Node.js
XSL
WSDL

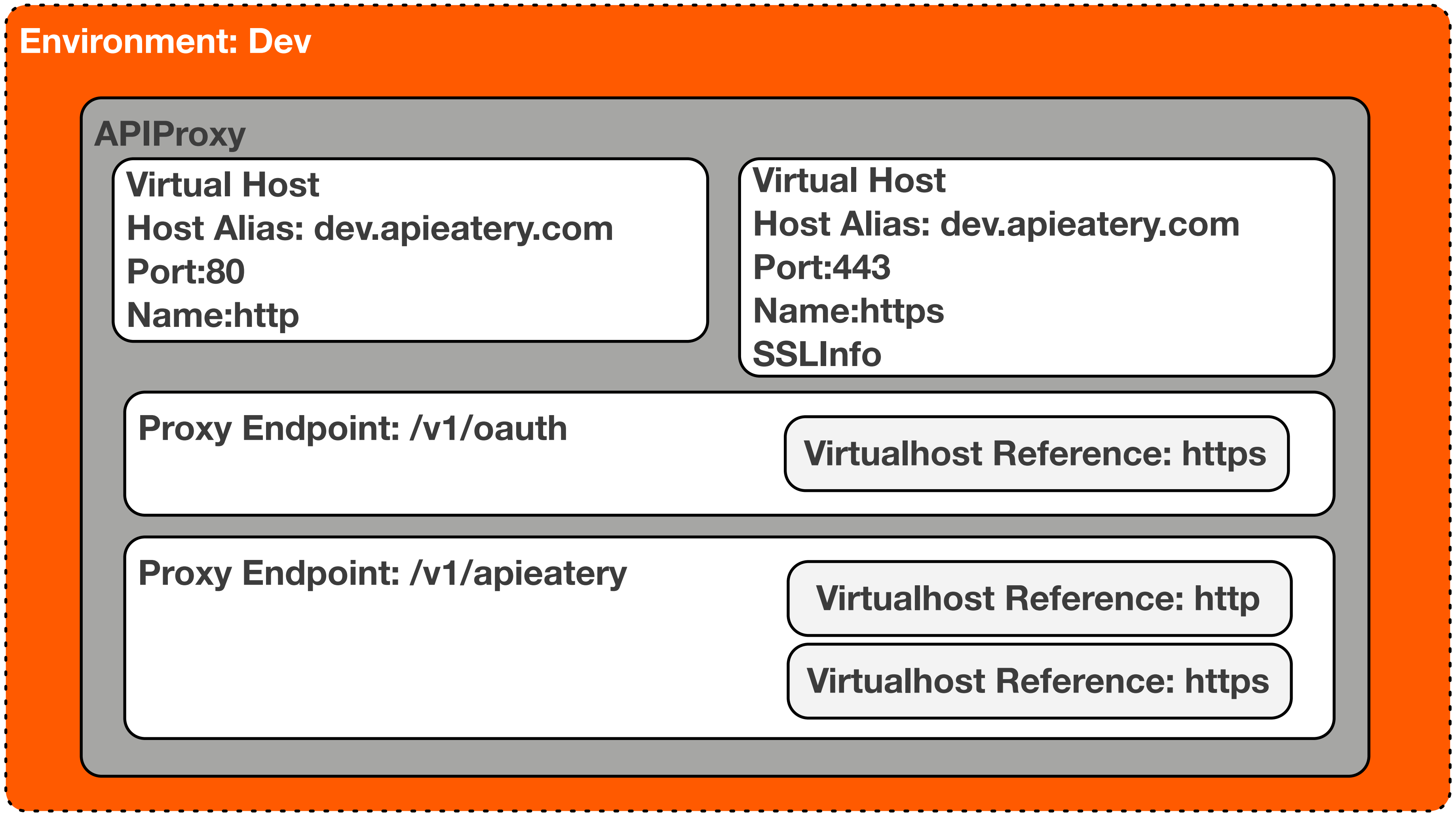
Proxy Endpoint



Proxy Endpoint Example

```xml
<ProxyEndpoint name="default">
    <DefaultFaultRule name="generic_fault_handler">
        <Step><Name>fault_genericfault</Name></Step>
        <AlwaysEnforce>true</AlwaysEnforce>
    </DefaultFaultRule>
    <FaultRules>
        <FaultRule name="invalid_key_rule">
            <Step><Name>fault_invalidkey</Name></Step>
            <Condition>fault.name = "InvalidApiKey"</Condition>
        </FaultRule>
    </FaultRules>
    <Flows/>
    <PreFlow name="PreFlow"/>
    <PostFlow name="PostFlow"/>
    <HTTPProxyConnection>
        <BasePath>/v1/someAPI</BasePath>
        <VirtualHost>default</VirtualHost>
    </HTTPProxyConnection>
    <RouteRule name="default">
        <TargetEndpoint>default</TargetEndpoint>
    </RouteRule>
</ProxyEndpoint>
```

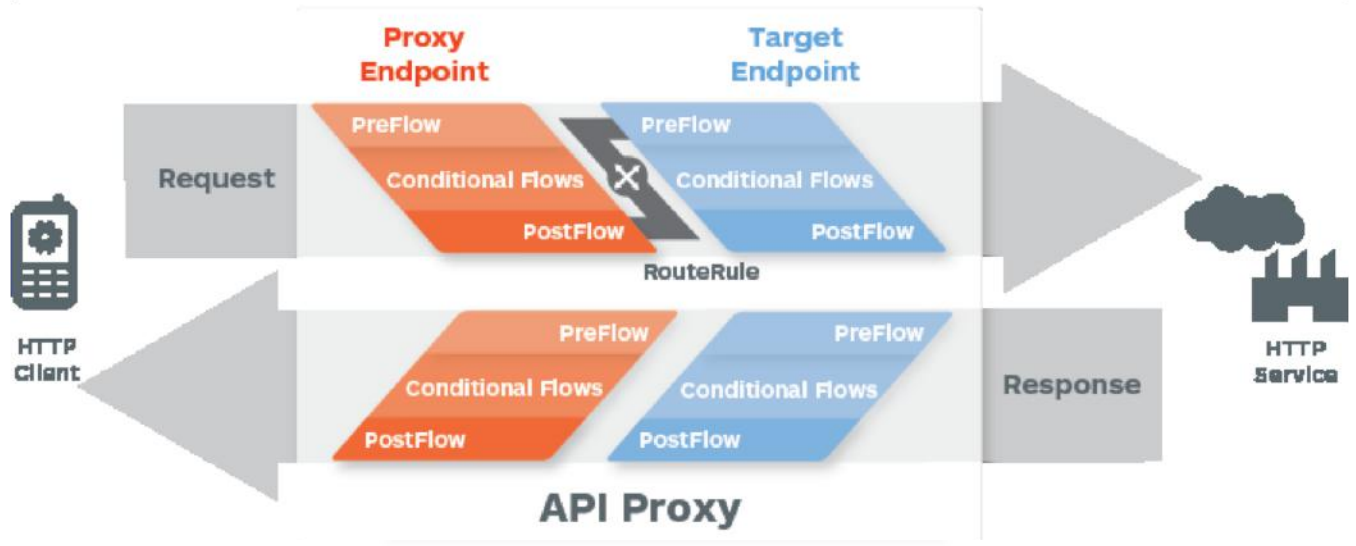
Virtual Hosts


Environment: Dev
APIProxy
Virtual Host
Host Alias: dev.apieatery.com
Port:80
Name:http
Virtual Host
Host Alias: dev.apieatery.com
Port:443
Name:https
SSLInfo
Proxy Endpoint: /v1/oauth
Virtualhost Reference: https
Proxy Endpoint: /v1/apieatery
Virtualhost Reference: http
Virtualhost Reference: https

Virtual Hosts

```xml
<HTTPProxyConnection>
    <BasePath>/v1/someAPI</BasePath>
    <VirtualHost>default</VirtualHost>
    <VirtualHost>secure</VirtualHost>
    <VirtualHost>beta</VirtualHost>
</HTTPProxyConnection>
```

Policy Flows



apiproxy/proxies/default.xml

```
<ProxyEndpoint name="default">
  <PreFlow>
    <Request>1</Request>
    <Response>10</Response>
  </PreFlow>
  <Flows>
    <Flow name="getDogs">
      <Request>2</Request>
      <Response>11</Response>
      <Condition></Condition>
    </Flow>
  </Flows>
  <PostFlow>
    <Request>3</Request>
    <Response>12</Response>
  </PostFlow>
  ...
```

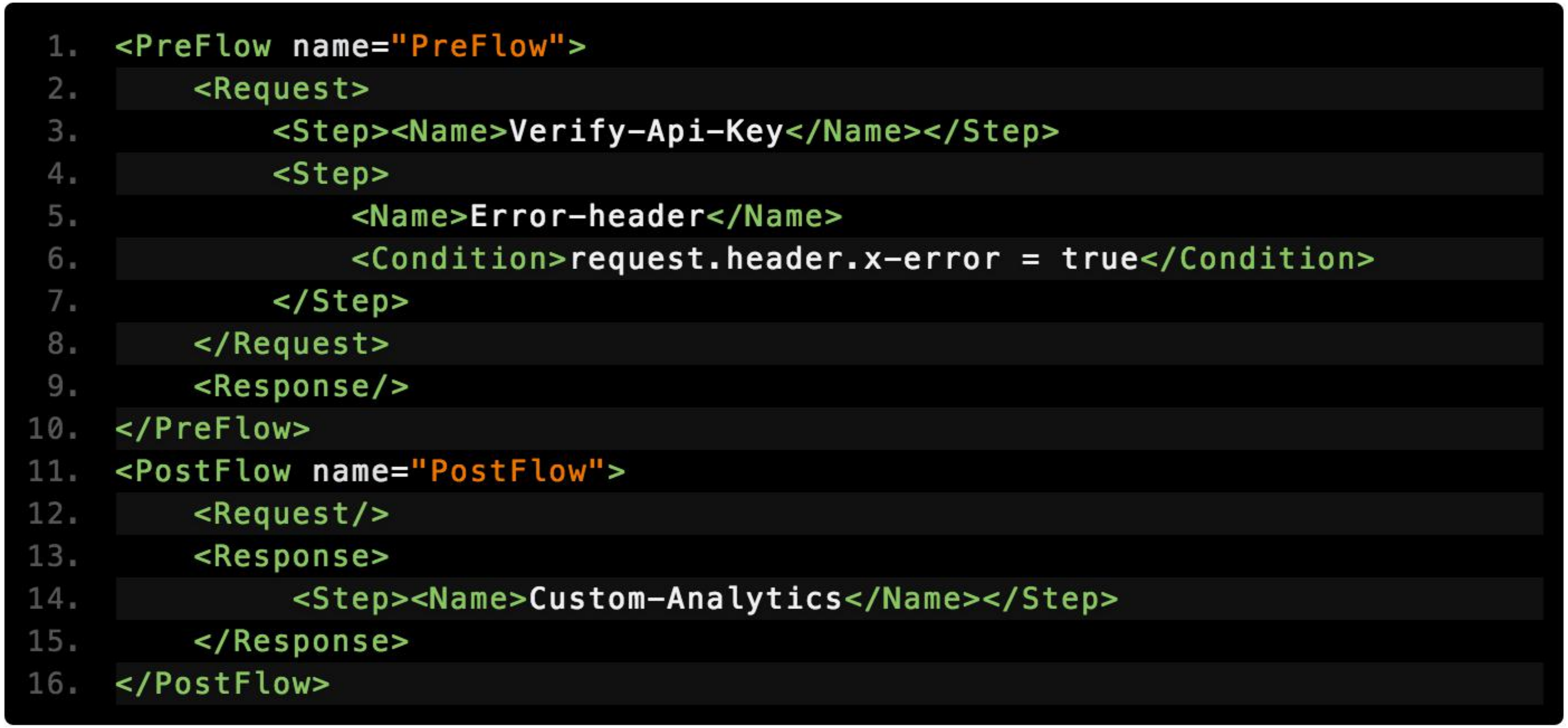
apiproxy/targets/default.xml

```
<TargetEndpoint name="default">
  <PreFlow>
    <Request>4</Request>
    <Response>7</Response>
  </PreFlow>
  <Flows>
    <Flow name="purchaseTarget">
      <Request>5</Request>
      <Response>8</Response>
      <Condition></Condition>
    </Flow>
  </Flows>
  <PostFlow>
    <Request>6</Request>
    <Response>9</Response>
  </PostFlow>
  ...
```

apigee

Policy Flows

Global Flows



```
<PreFlow name="PreFlow">
    <Request>
        <Step><Name>Verify-Api-Key</Name></Step>
        <Step>
            <Name>Error-header</Name>
            <Condition>request.header.x-error = true</Condition>
        </Step>
    </Request>
    <Response/>
</PreFlow>
<PostFlow name="PostFlow">
    <Request/>
    <Response>
         <Step><Name>Custom-Analytics</Name></Step>
    </Response>
</PostFlow>
```

apigee

Policy Flows

Conditional Flows / Resource

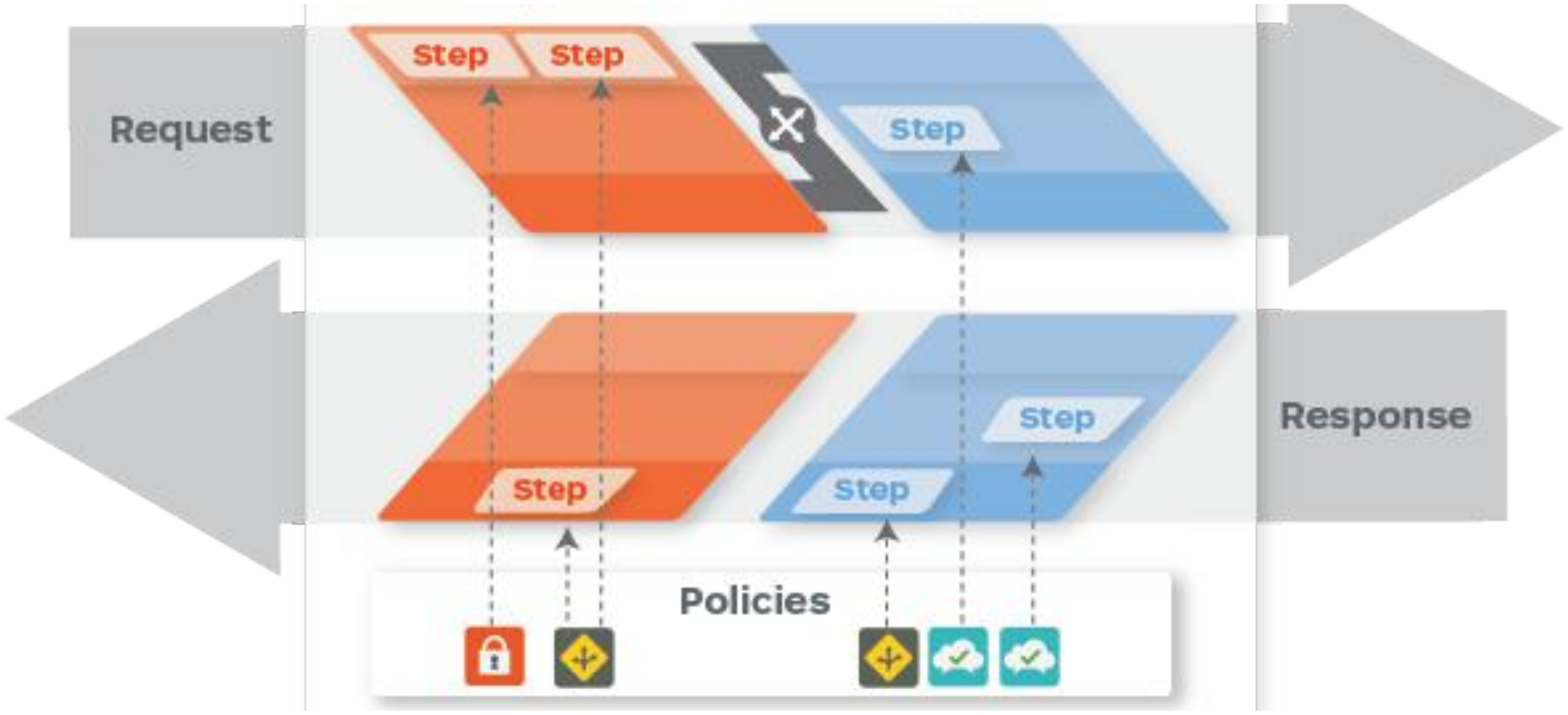
```
<Flows>
    <Flow name="GetResource">
        <Description>Get Resource</Description>
        <Request>
            <Step><Name>Resource-Quota</Name></Step>
        </Request>
        <Response/>
        <Condition>(proxy.pathsuffix MatchesPath "resource") and (request.verb = "GET")</Condition>
    </Flow>
</Flows>
```

Post Client

```
<PostClientFlow>
     <Response>
         <Step><Name>Send-Audit-Info</Name></Step>
    </Response>
</PostClientFlow>
```

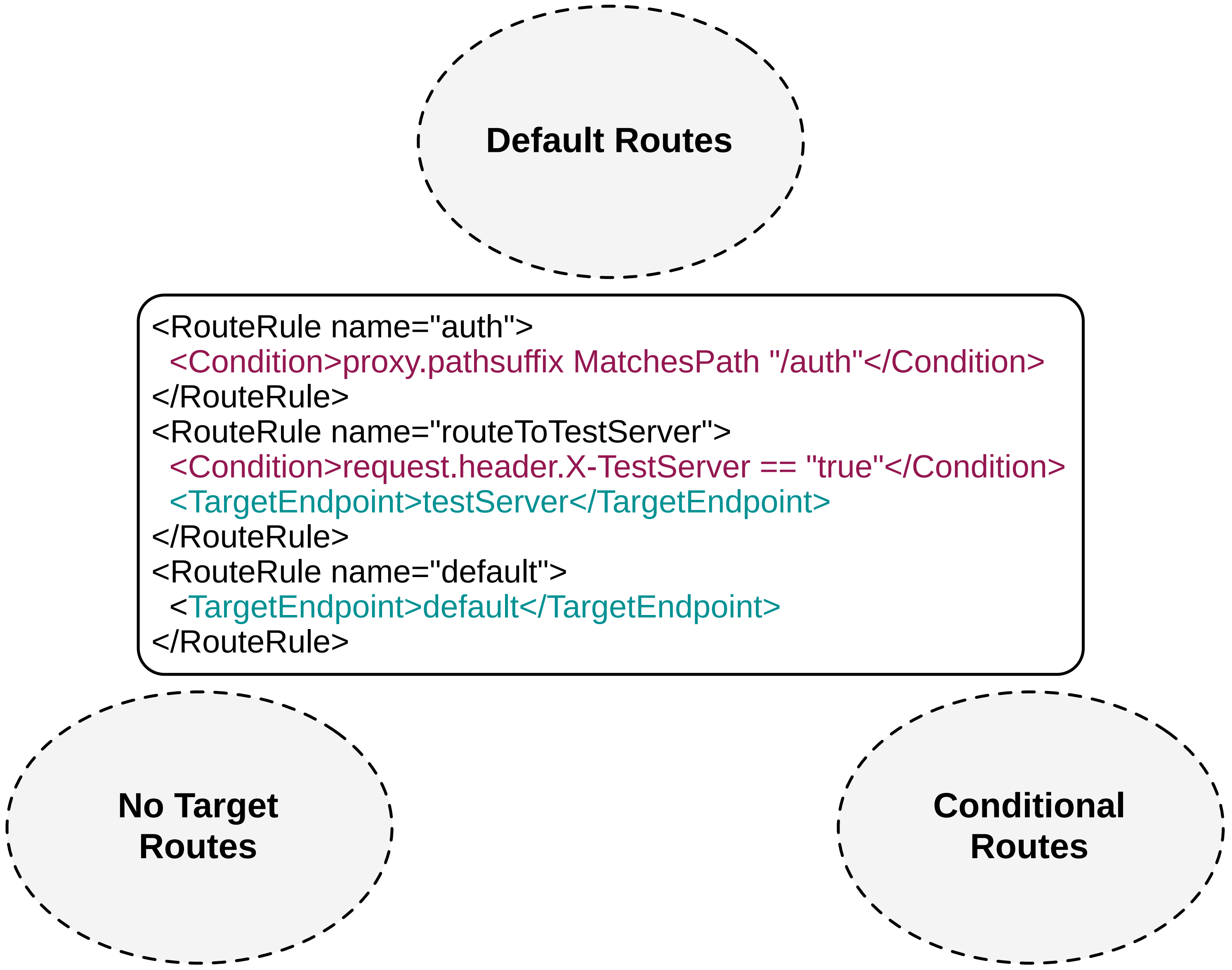
apigee

Policies



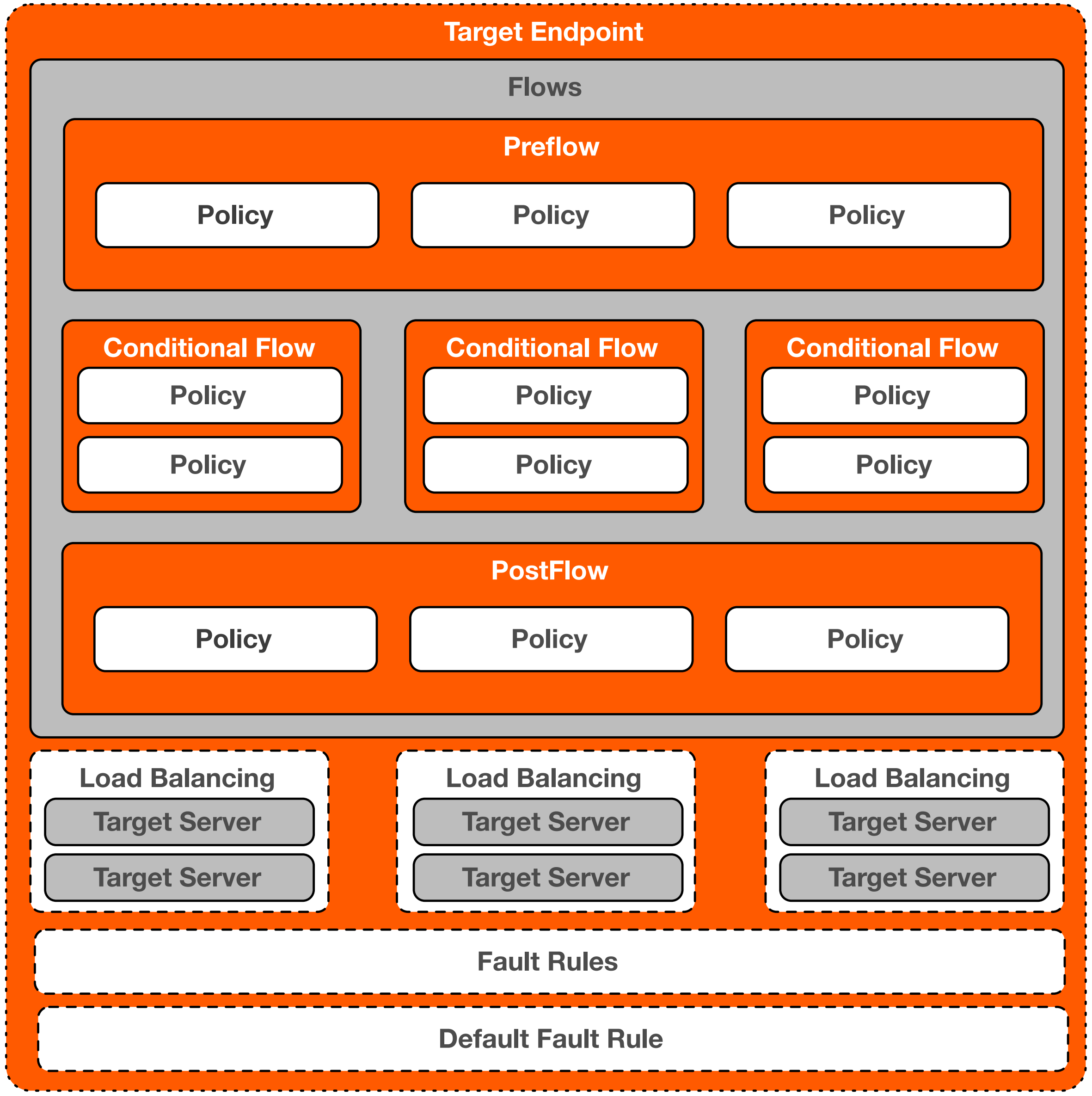
apigee

Route Rules

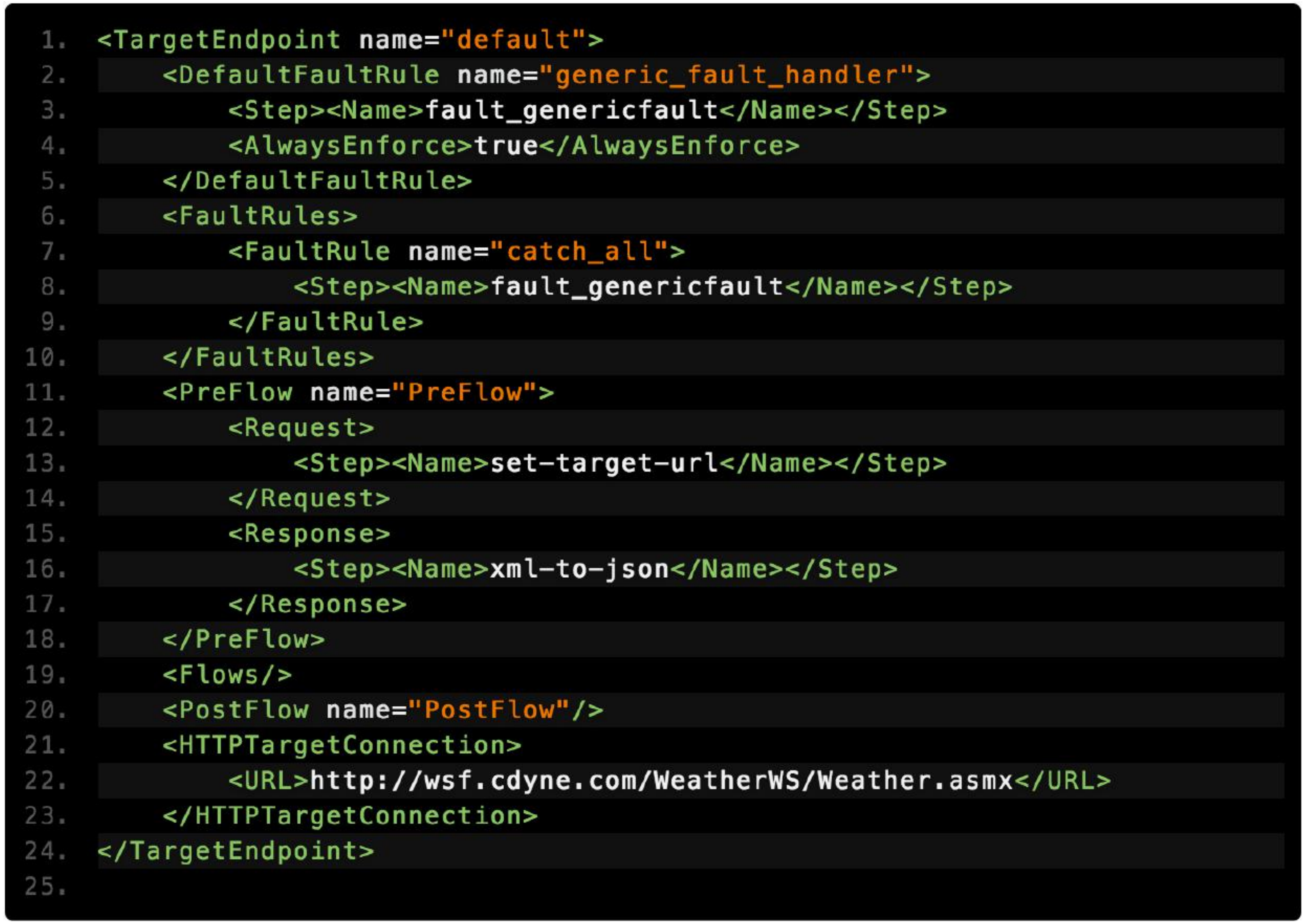


```
<RouteRule name="auth">
  <Condition>proxy.pathsuffix MatchesPath "/auth"</Condition>
</RouteRule>
<RouteRule name="routeToTestServer">
  <Condition>request.header.X-TestServer == "true"</Condition>
  <TargetEndpoint>testServer</TargetEndpoint>
</RouteRule>
<RouteRule name="default">
  <TargetEndpoint>default</TargetEndpoint>
</RouteRule>
```

Target Endpoints



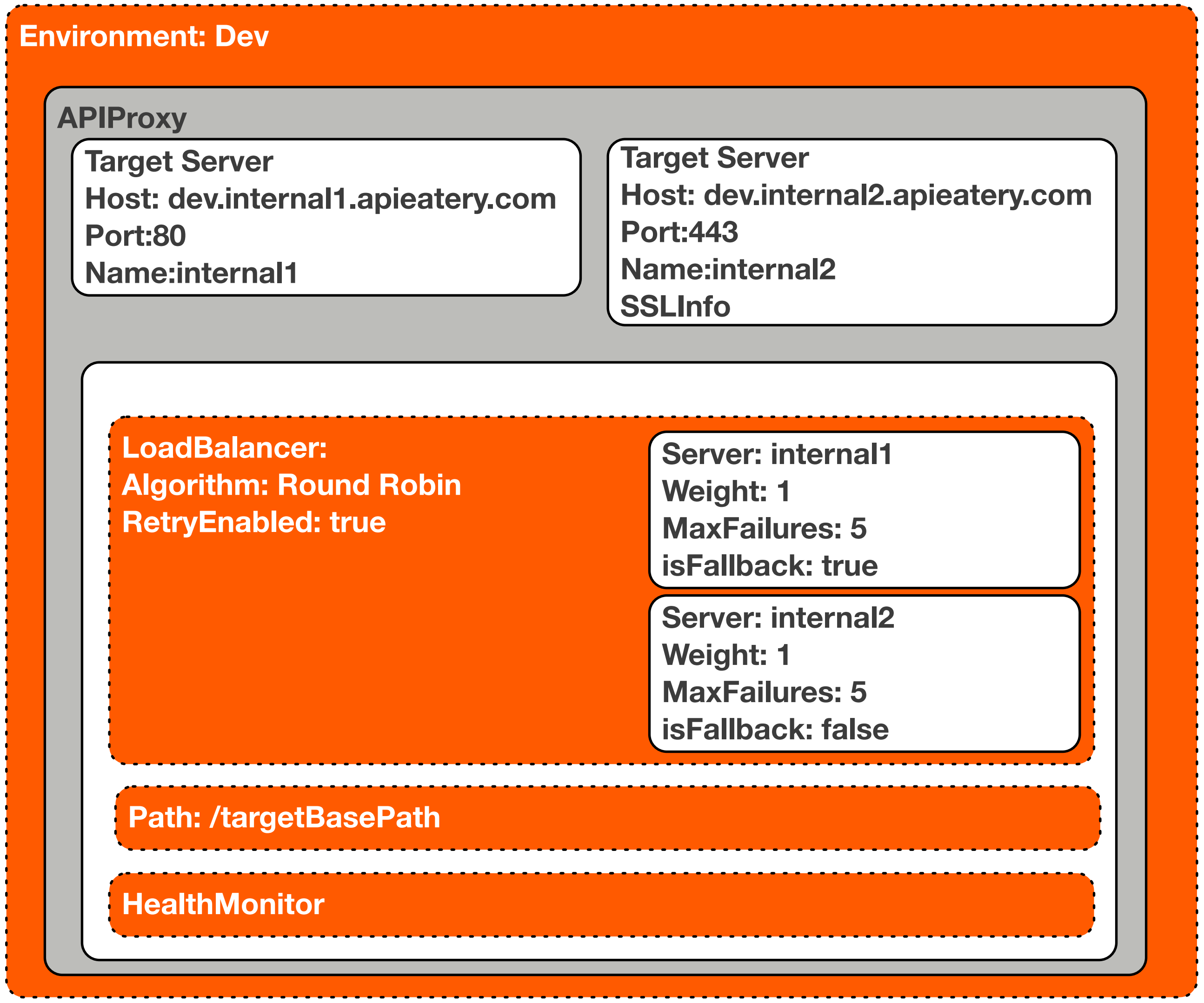
Target Endpoints



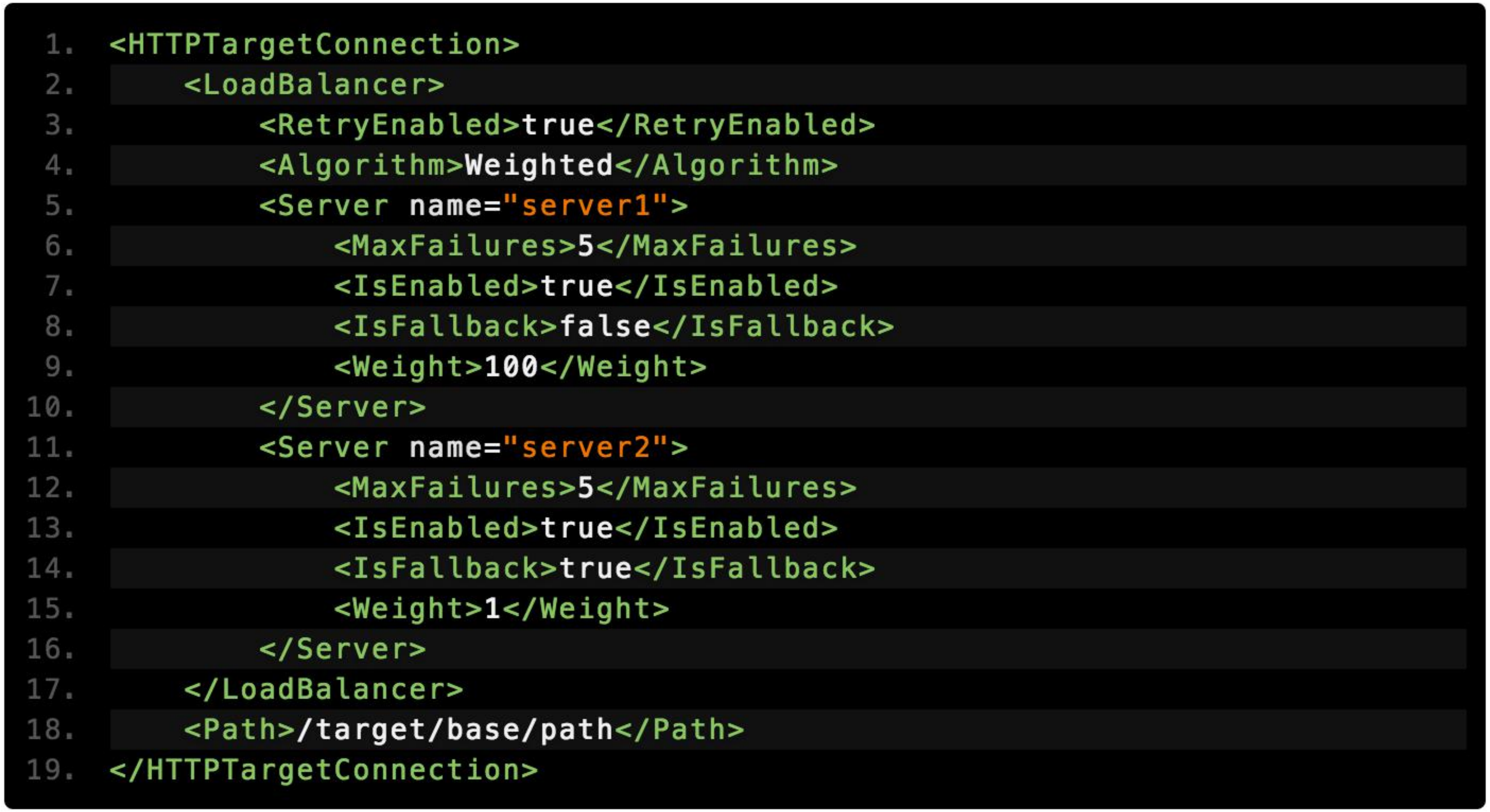
```xml
<TargetEndpoint name="default">
    <DefaultFaultRule name="generic_fault_handler">
        <Step><Name>fault_genericfault</Name></Step>
        <AlwaysEnforce>true</AlwaysEnforce>
    </DefaultFaultRule>
    <FaultRules>
        <FaultRule name="catch_all">
            <Step><Name>fault_genericfault</Name></Step>
        </FaultRule>
    </FaultRules>
    <PreFlow name="PreFlow">
        <Request>
            <Step><Name>set-target-url</Name></Step>
        </Request>
        <Response>
            <Step><Name>xml-to-json</Name></Step>
        </Response>
    </PreFlow>
    <Flows/>
    <PostFlow name="PostFlow"/>
    <HTTPTargetConnection>
        <URL>http://wsf.cdyne.com/WeatherWS/Weather.asmx</URL>
    </HTTPTargetConnection>
</TargetEndpoint>

```

apigee

Load Balancing

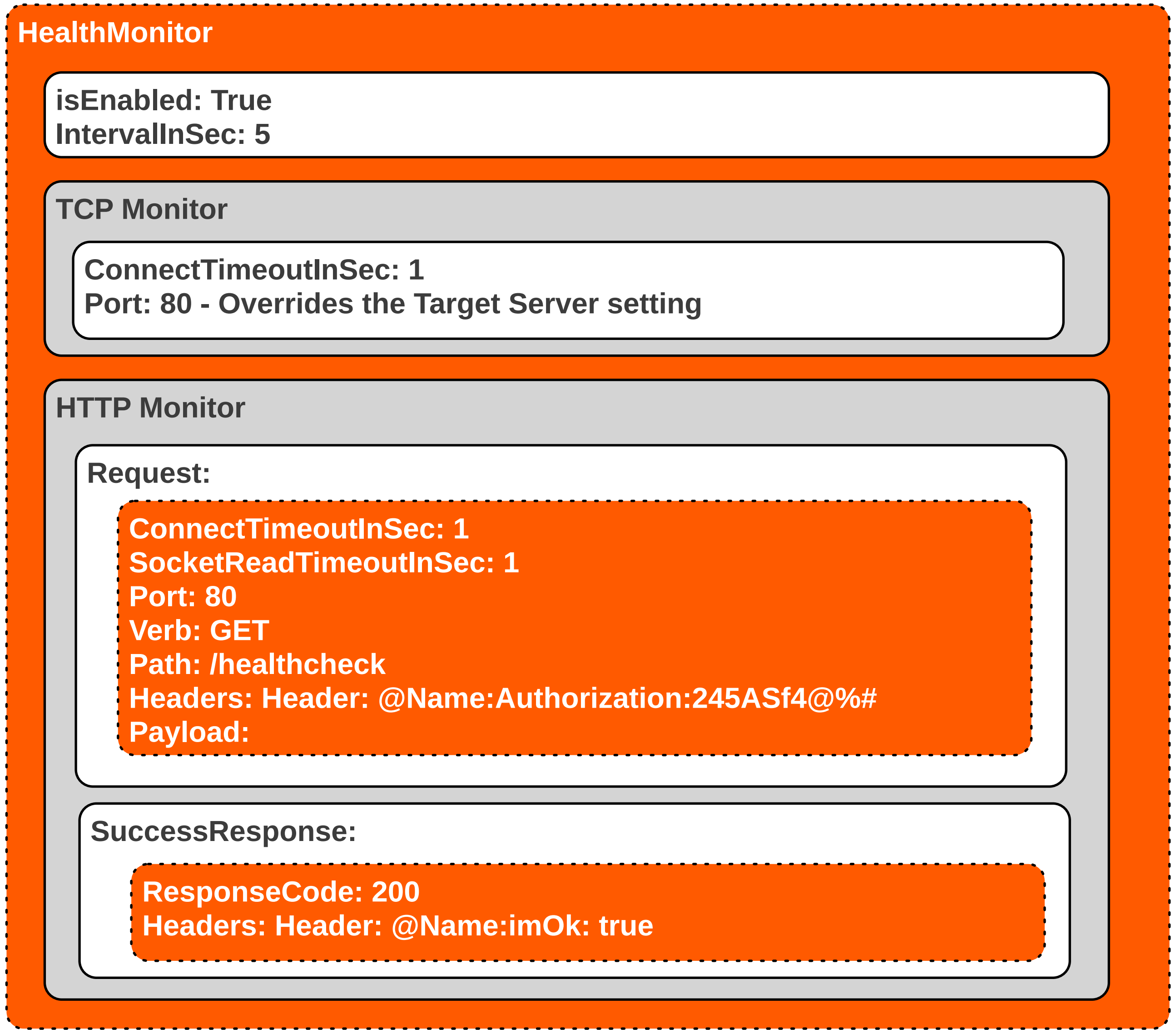


Load Balancer



```xml
<HTTPTargetConnection>
    <LoadBalancer>
        <RetryEnabled>true</RetryEnabled>
        <Algorithm>Weighted</Algorithm>
        <Server name="server1">
            <MaxFailures>5</MaxFailures>
            <IsEnabled>true</IsEnabled>
            <IsFallback>false</IsFallback>
            <Weight>100</Weight>
        </Server>
        <Server name="server2">
            <MaxFailures>5</MaxFailures>
            <IsEnabled>true</IsEnabled>
            <IsFallback>true</IsFallback>
            <Weight>1</Weight>
        </Server>
    </LoadBalancer>
    <Path>/target/base/path</Path>
</HTTPTargetConnection>
```

Health Monitor



HealthMonitor

isEnabled: True
IntervalInSec: 5

TCP Monitor

ConnectTimeoutInSec: 1
Port: 80 - Overrides the Target Server setting

HTTP Monitor

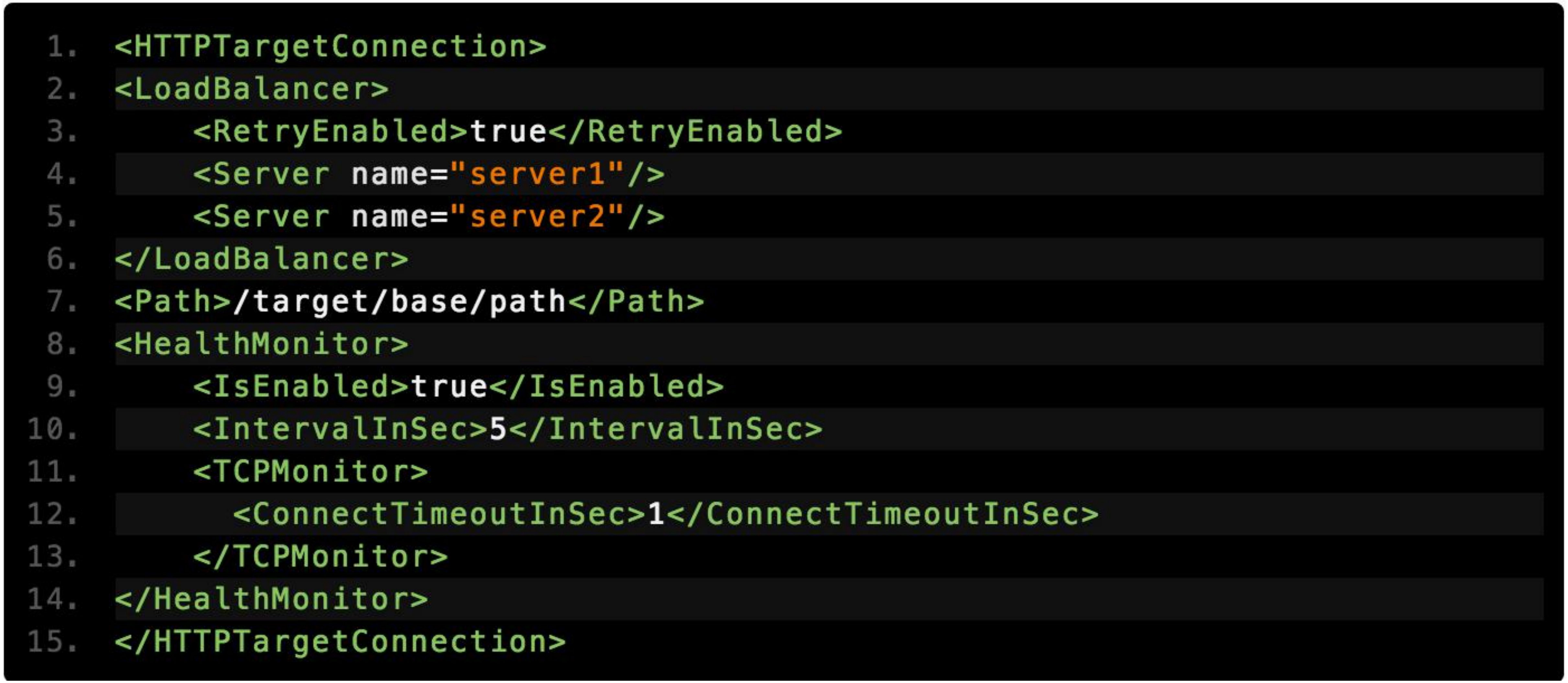
Request:

ConnectTimeoutInSec: 1
SocketReadTimeoutInSec: 1
Port: 80
Verb: GET
Path: /healthcheck
Headers: Header: @Name:Authorization:245ASf4@%#
Payload:

SuccessResponse:

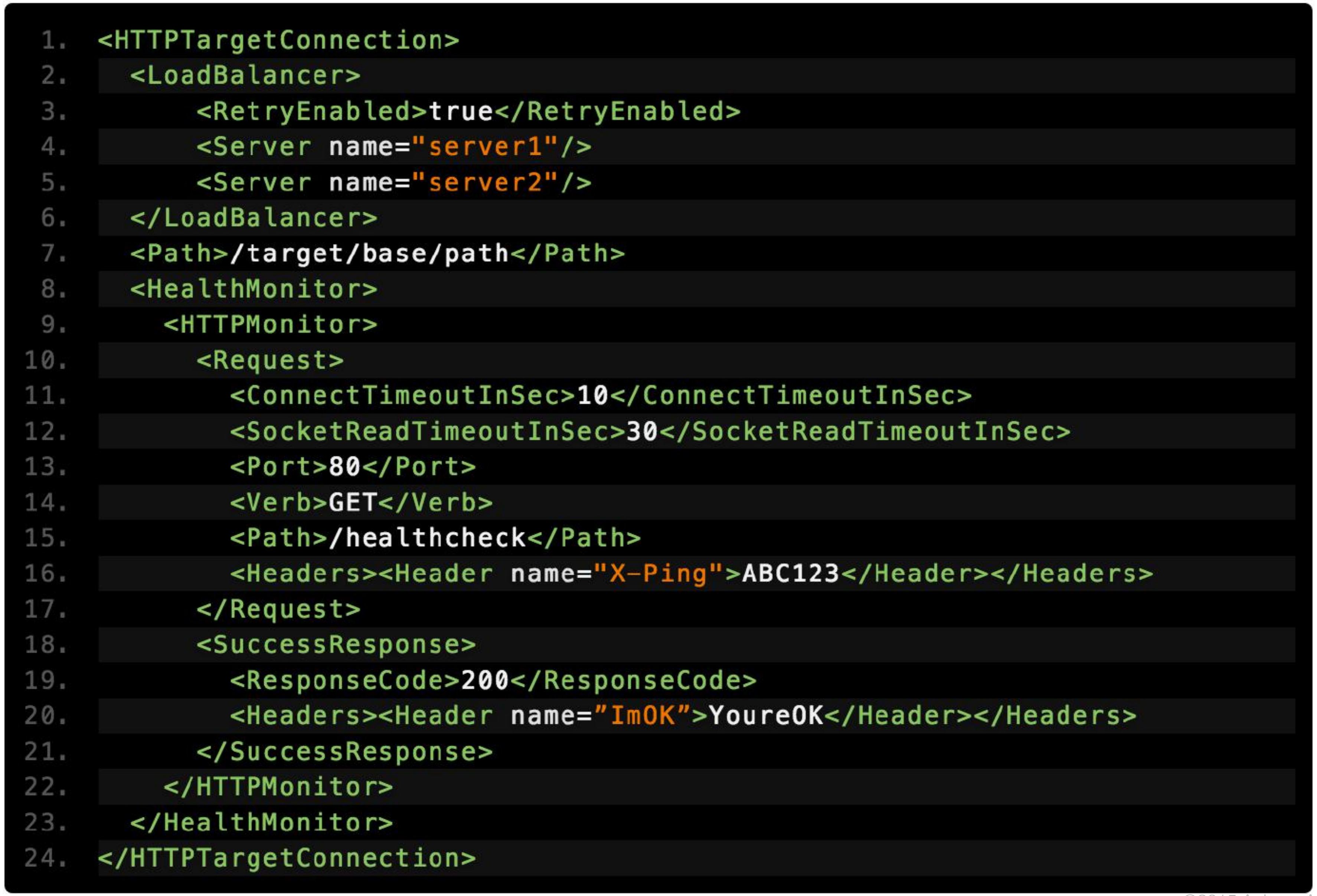
ResponseCode: 200
Headers: Header: @Name:imOk: true

TCP Health Checks



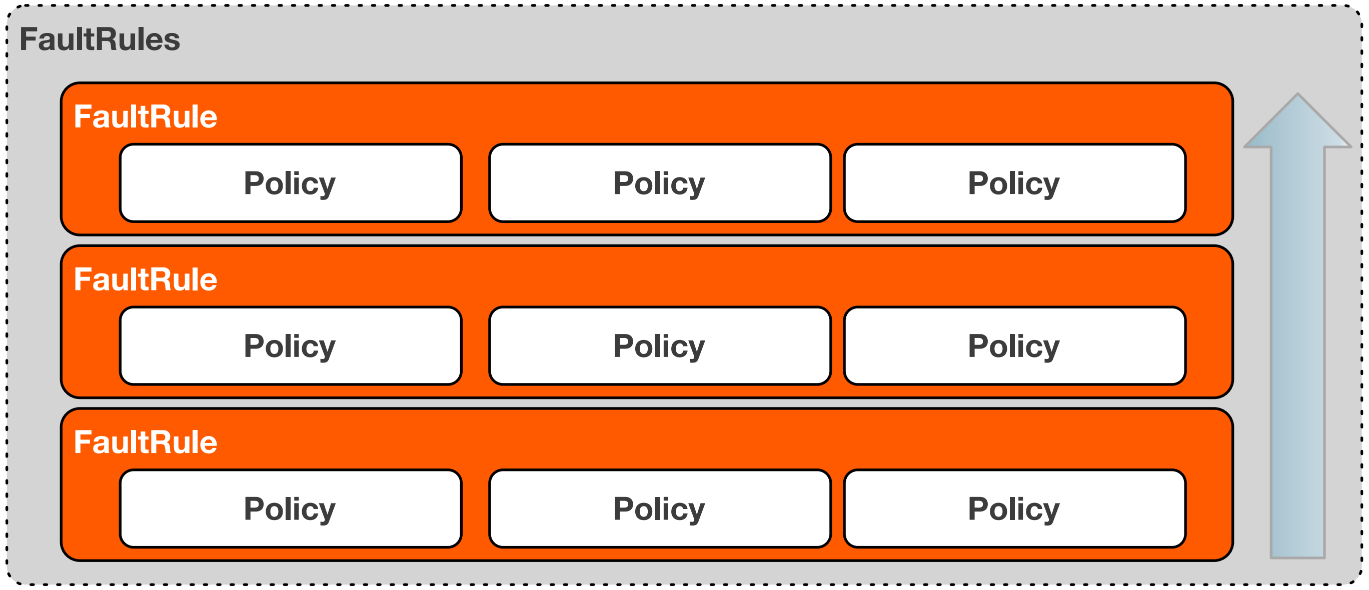
```xml
<HTTPTargetConnection>
<LoadBalancer>
    <RetryEnabled>true</RetryEnabled>
    <Server name="server1"/>
    <Server name="server2"/>
</LoadBalancer>
<Path>/target/base/path</Path>
<HealthMonitor>
    <IsEnabled>true</IsEnabled>
    <IntervalInSec>5</IntervalInSec>
    <TCPMonitor>
      <ConnectTimeoutInSec>1</ConnectTimeoutInSec>
    </TCPMonitor>
</HealthMonitor>
</HTTPTargetConnection>
```

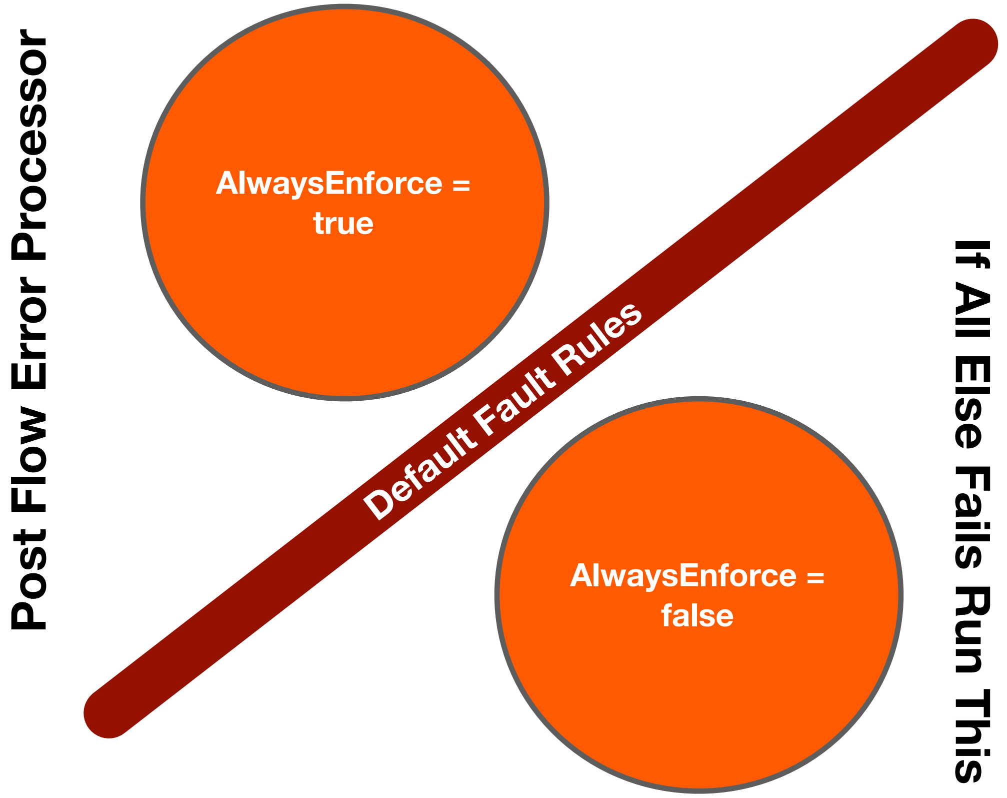
HTTP Health Checks



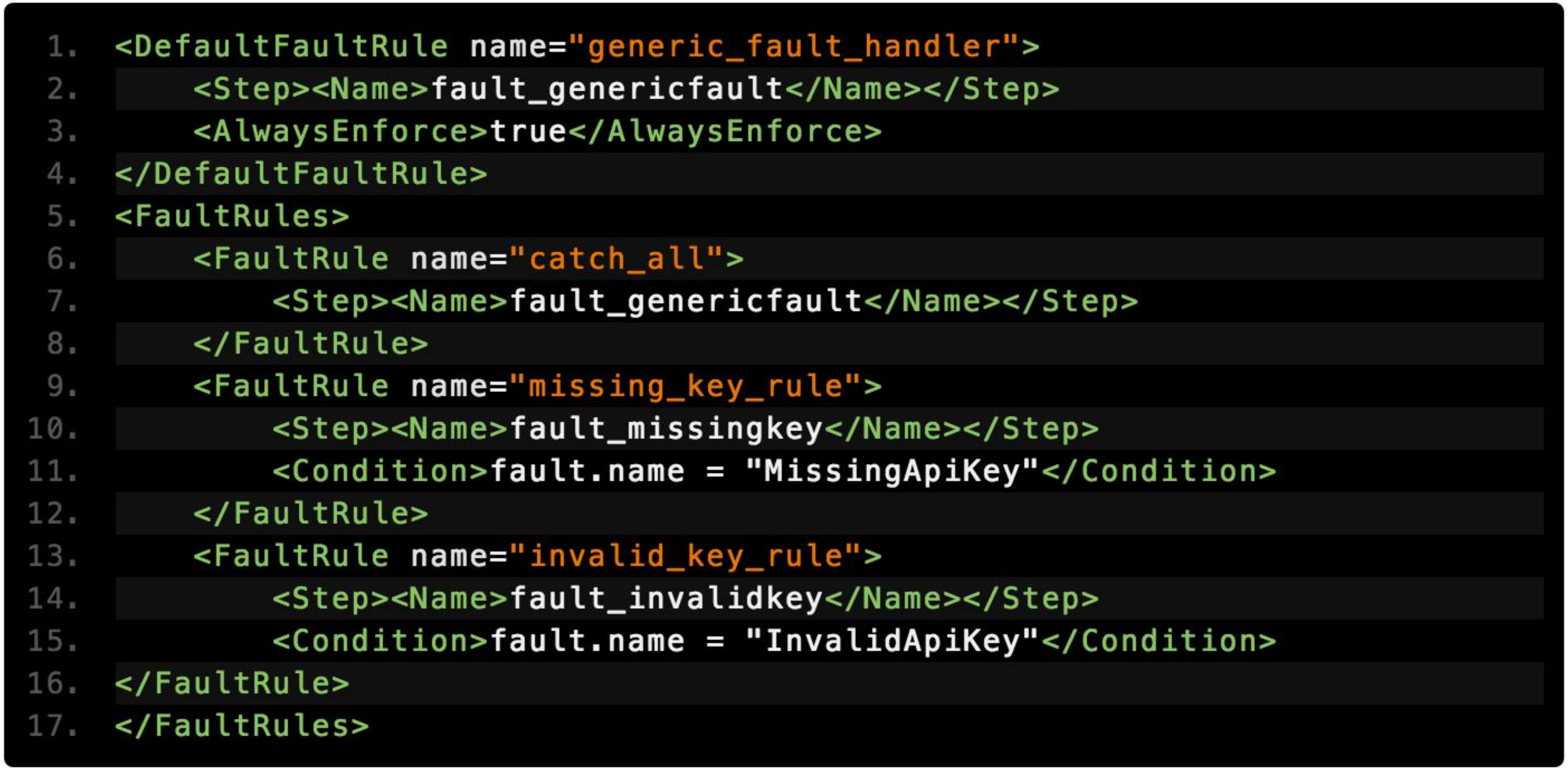
```xml
<HTTPTargetConnection>
  <LoadBalancer>
      <RetryEnabled>true</RetryEnabled>
      <Server name="server1"/>
      <Server name="server2"/>
  </LoadBalancer>
  <Path>/target/base/path</Path>
  <HealthMonitor>
    <HTTPMonitor>
      <Request>
        <ConnectTimeoutInSec>10</ConnectTimeoutInSec>
        <SocketReadTimeoutInSec>30</SocketReadTimeoutInSec>
        <Port>80</Port>
        <Verb>GET</Verb>
        <Path>/healthcheck</Path>
        <Headers><Header name="X-Ping">ABC123</Header></Headers>
      </Request>
      <SuccessResponse>
        <ResponseCode>200</ResponseCode>
        <Headers><Header name=”ImOK”>YoureOK</Header></Headers>
      </SuccessResponse>
    </HTTPMonitor>
  </HealthMonitor>
</HTTPTargetConnection>
```

Fault Rules


FaultRules
FaultRule
Policy
Policy
Policy
FaultRule
Policy
Policy
Policy
FaultRule
Policy
Policy
Policy


Post Flow Error Processor
AlwaysEnforce = true
Default Fault Rules
AlwaysEnforce = false
If All Else Fails Run This

Fault Rules



```
<DefaultFaultRule name="generic_fault_handler">
    <Step><Name>fault_genericfault</Name></Step>
    <AlwaysEnforce>true</AlwaysEnforce>
</DefaultFaultRule>
<FaultRules>
    <FaultRule name="catch_all">
        <Step><Name>fault_genericfault</Name></Step>
    </FaultRule>
    <FaultRule name="missing_key_rule">
        <Step><Name>fault_missingkey</Name></Step>
        <Condition>fault.name = "MissingApiKey"</Condition>
    </FaultRule>
    <FaultRule name="invalid_key_rule">
        <Step><Name>fault_invalidkey</Name></Step>
        <Condition>fault.name = "InvalidApiKey"</Condition>
</FaultRule>
</FaultRules>
```

Advanced Endpoint Properties For Proxy

In addition to the standard endpoint configurations there are some properties to control advanced functionality.

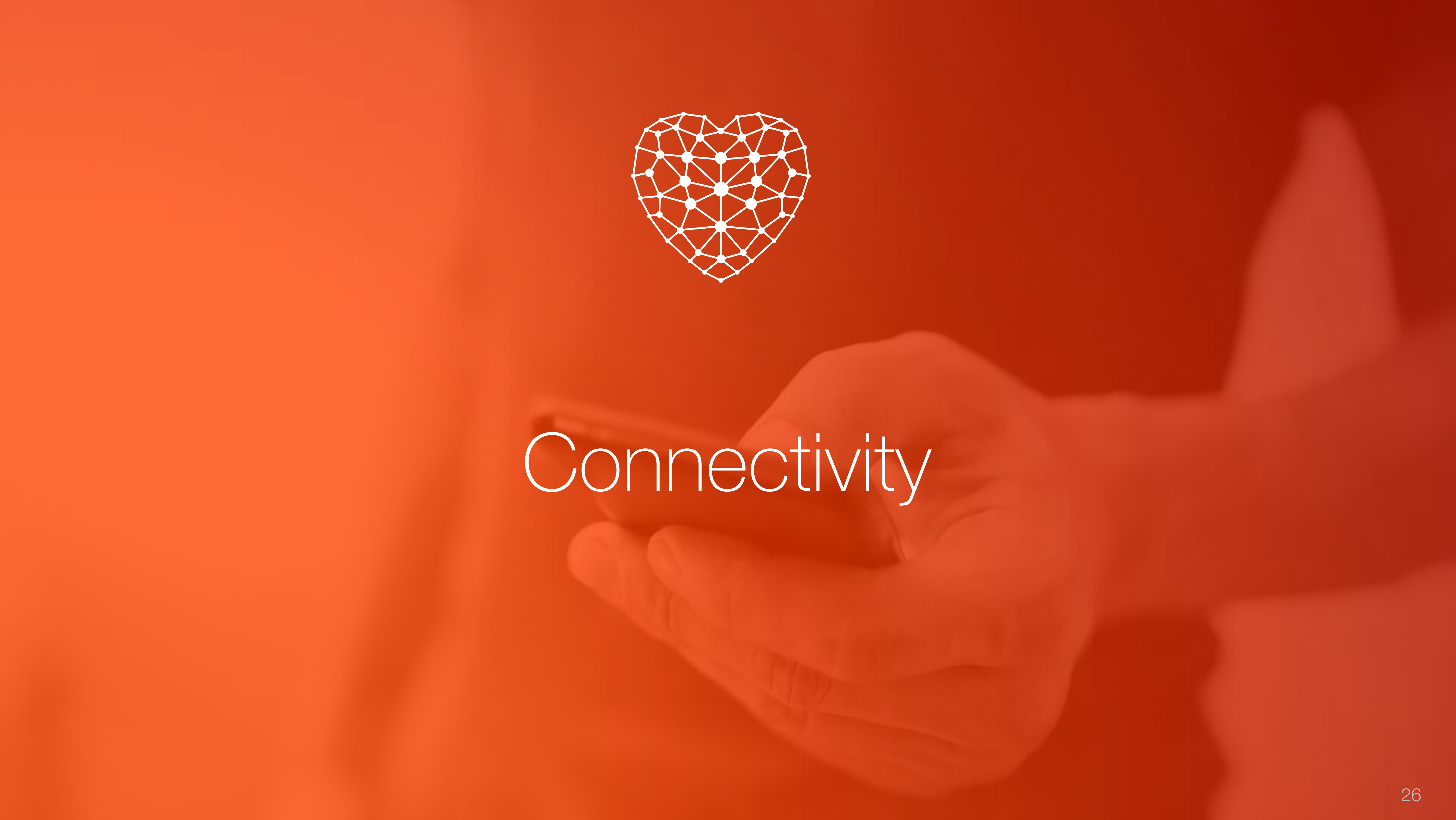
<HTTPProxyConnection>

- **allow.http.method.*** - by default all HTTP methods are allowed. To disable a method you can use this property: <Property name="allow.http.method.POST">false</Property>
- **request.streaming.enabled** – if the payload will not be read or updated during the proxy processing, you can enable to to avoid payload buffer size limits
- **response.streaming.enabled** - if the payload will not be read or updated during the proxy processing, you can enable to to avoid payload buffer size limits

Advanced Endpoint Properties For Target

<HTTPTargetConnection>

- **connect.timeout.millis** – this is used to set the connection timeout for backend connections (in ms). Defaults to 60000 (1 min) and highly recommended to lower
- **io.timeout.millis** – this is used to set the response read timeout (waiting for response from backend). Defaults to 120000 (2 min) and extremely encouraged to reduce to optimize connection handling in the event that a backend service is degraded.
- **response.streaming.enabled** - if the payload will not be read or updated during the proxy processing, you can enable to to avoid payload buffer size limits
- **success.codes** – this property is used to set the response status codes that are treated as 'success'. Defaults to 1XX,2XX,3XX. This is a helpful configuration when you want to access 4XX responses are success to continue response flow processing



Connectivity

Getting Setup

Steps

1. Using a web browser Navigate to apigee.com
2. Click the Sign In Button
3. If you do not have an Apigee Account click the Sign Up Link
4. Fill out form and click the Create Account button
5. Activate Account by clicking link contained in an email sent to your registered account
6. Once your account is activated you should be able to login, once logged in you will be taken to the dashboard page, once their click the button under the API Management block.

Note: Activation can sometimes take a couple of minutes to activate.

Welcome to Apigee Edge!!!!!

New API Proxy Options Part 1

New API Proxy Options

- Backend Service
  - Most common starting point when building a proxy from scratch
  - Backend Service URL field is used to target a specific backend endpoint
- API Bundle
  - This Option allows you to define an existing API proxy that is bundled in a ZIP file format for import.
  - Commonly used to import a proxy from one organization to another
- WSDL
  - This option is used for creating policies that work with an existing web service along with your new proxy
  - Supports Pass-Thru options
  - Supports Basic Rest too Soap Conversions

New API Proxy Options Part 2

New API Proxy Options

- No Target
  - This option is used when you service the API directly on the Apigee Platform with no Target
  - Is great for creating API Stubs
- Node.js
  - Two Options for Node.js (New, Existing)
  - The New option gives you the capability of creating a new API proxy that has a simple Node.js Hello World Application built into it (there is also an example of a BAAS target too).
  - The Existing option is like the API Bundle option, the only difference is that it only excepts your entry js file e.g. app.js or server .js

Completing the Form

Lets Complete the Form

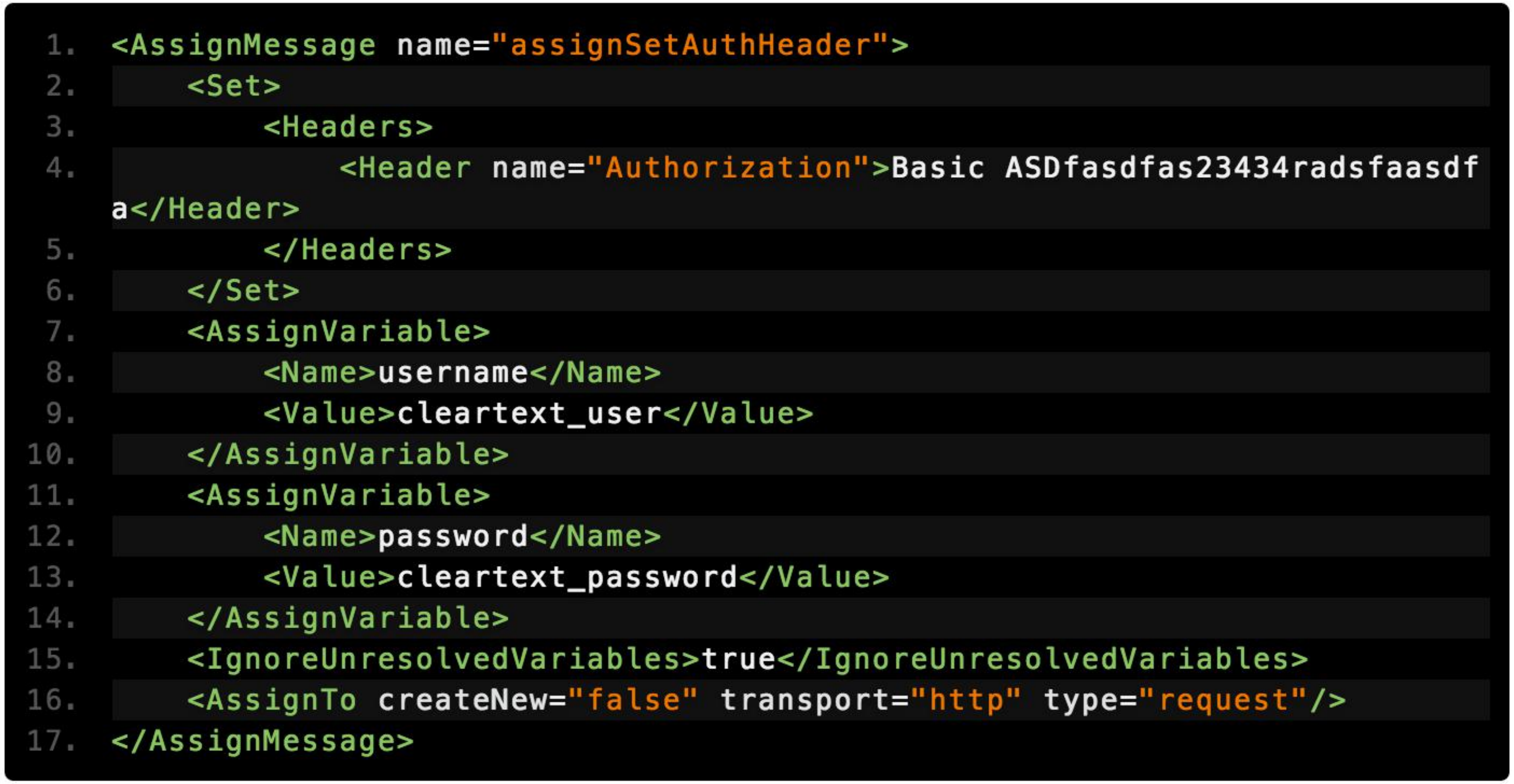
- Use Backend Service
  - Set the **Backend Service URL** to:
    - https://apigee-edu-test.apigee.net/v1/apieatery
  - In the **Name** use something unique to you, for instance yourname_apieatery
  - In the **Project Base Path**: add a path to make it unique I use /v1/myname/apieatery

Building Target Authentication

Building our Proxy against a target that uses Basic Authentication requires you to store credentials on the Apigee Platform. Which poses the question "Were do i store the credentials?”

1. Hard Code Them?
2. Replace Them at Build Time?
3. Key Value Map?
4. Node.js Vault?

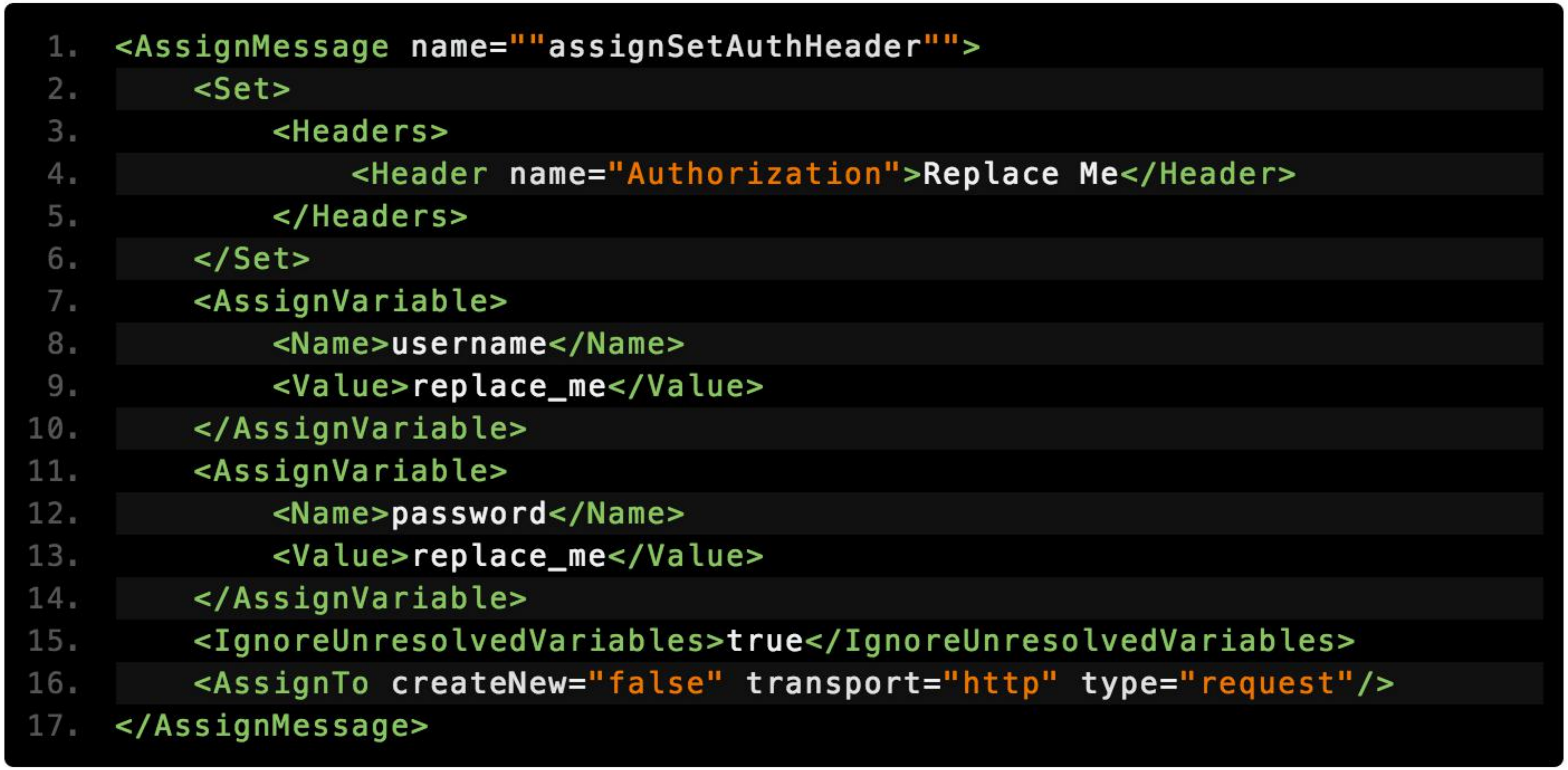
Hard Coding using Assign Message



```xml
<AssignMessage name="assignSetAuthHeader">
    <Set>
        <Headers>
            <Header name="Authorization">Basic ASDfasdfas23434radsfaasdfa</Header>
        </Headers>
    </Set>
    <AssignVariable>
        <Name>username</Name>
        <Value>cleartext_user</Value>
    </AssignVariable>
    <AssignVariable>
        <Name>password</Name>
        <Value>cleartext_password</Value>
    </AssignVariable>
    <IgnoreUnresolvedVariables>true</IgnoreUnresolvedVariables>
    <AssignTo createNew="false" transport="http" type="request"/>
</AssignMessage>
```

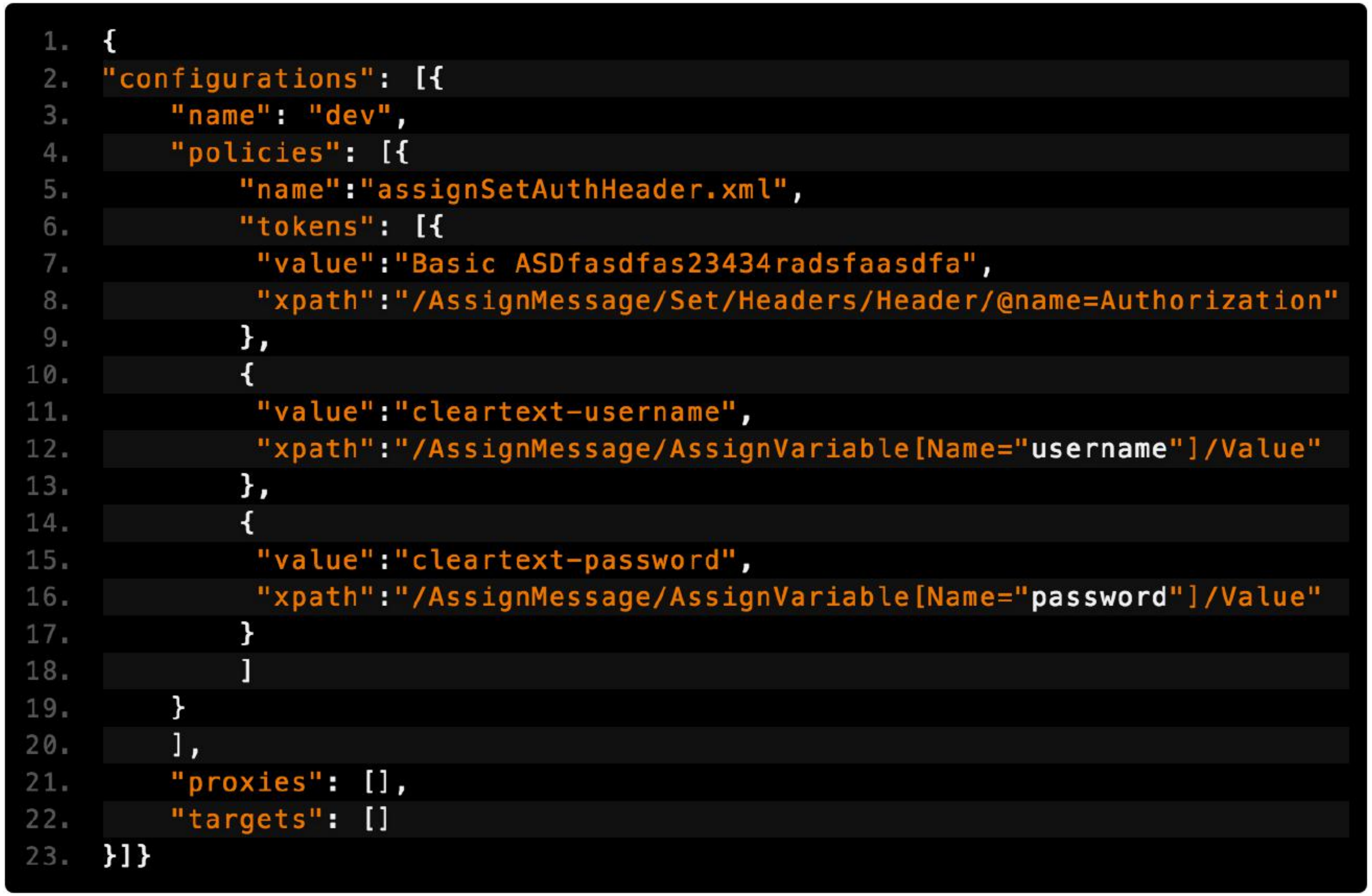
apigee

Build Time Replacement



```xml
<AssignMessage name=""assignSetAuthHeader"">
    <Set>
        <Headers>
            <Header name="Authorization">Replace Me</Header>
        </Headers>
    </Set>
    <AssignVariable>
        <Name>username</Name>
        <Value>replace_me</Value>
    </AssignVariable>
    <AssignVariable>
        <Name>password</Name>
        <Value>replace_me</Value>
    </AssignVariable>
    <IgnoreUnresolvedVariables>true</IgnoreUnresolvedVariables>
    <AssignTo createNew="false" transport="http" type="request"/>
</AssignMessage>
```

Build Time Replacement Part 2



```
{
"configurations": [{
    "name": "dev",
    "policies": [{
        "name":"assignSetAuthHeader.xml",
        "tokens": [{
         "value":"Basic ASDfasdfas23434radsfaasdfa",
         "xpath":"/AssignMessage/Set/Headers/Header/@name=Authorization"
        },
        {
         "value":"cleartext-username",
         "xpath":"/AssignMessage/AssignVariable[Name="username"]/Value"
        },
        {
         "value":"cleartext-password",
         "xpath":"/AssignMessage/AssignVariable[Name="password"]/Value"
        }
        ]
    }
    ],
    "proxies": [],
    "targets": []
}]}
```

Storing Credentials in a Node.js Access Vault

Step 1: Build Vault and Vault Data Via API

```
curl https://api.enterprise.apigee.com/v1/o/{org}/vaults
  -H "Content-Type: application/json"
  -d '{"name": "vault" }' -X POST

curl https://api.enterprise.apigee.com/v1/o/{org}/vaults/{vault}/entries
  -H "Content-Type: application/json"
  -d '{"name": "value1", "value": "verysecret" }' -X POST
```

Step 2: Retrieve Data in Node.js App

```
var apigee = require('apigee-access');
var orgVault = apigee.getVault('vault1', 'organization');
    orgVault.get('key1', function(err, secretValue) {
    // use the secret value here
});
```

Storing Credentials in a Key Value Map

Step 1: Create KVM and KVM Data Via API

```
curl https://api.enterprise.apigee.com/v1/o/{org}/keyvaluemaps
  -H "Content-Type: application/json"
  -d '{ "name" : "credsMap", "entry" : [
  {"name" : "username",
   "value" : "foundationUser"},
  {"name" : "password",
   "value" : "Test1234"
  }]}' -X POST
```

Step 2: Retrieve Data via KVM Policy

```
<KeyValueMapOperations name="getUrl" mapIdentifier="credsMap">
    <Scope>organization</Scope>
    <Get assignTo="username" index='1'>
      <Key><Parameter>username</Parameter></Key>
    </Get>
    <Get assignTo="password" index='1'>
      <Key><Parameter>password</Parameter></Key>
   </Get>
</KeyValueMapOperations>
```

apigee

Key KVM Policy Configuration Options

- **ExpiryTimeInSecs** - This setting is used to expire the cache not the entry
- **InitialEntries** - Allows you to pre-define values

```
<InitialEntries>
    <Entry>
        <Key>
            <Parameter>key_name_literal</Parameter>
        </Key>
        <Value>value_literal</Value>
    </Entry>
</InitialEntries>
```

- **Multiple Values** - KVM supports multiple values per name, you can access values using the index attribute of the Get tag

Building Authorization Headers with the Basic Auth Policy

- **Can Encode the two variables into a basic authentication header**
- **Can Decode an basic authentication header into two variables**

Example

```
<BasicAuthentication name="base64">
    <DisplayName>buildAuthHeader</DisplayName>
    <Operation>Encode</Operation>
    <IgnoreUnresolvedVariables>false</IgnoreUnresolvedVariables>
    <User ref="username" />
    <Password ref="password" />
    <AssignTo createNew="true">request.header.Authorization</AssignTo>
</BasicAuthentication>
```

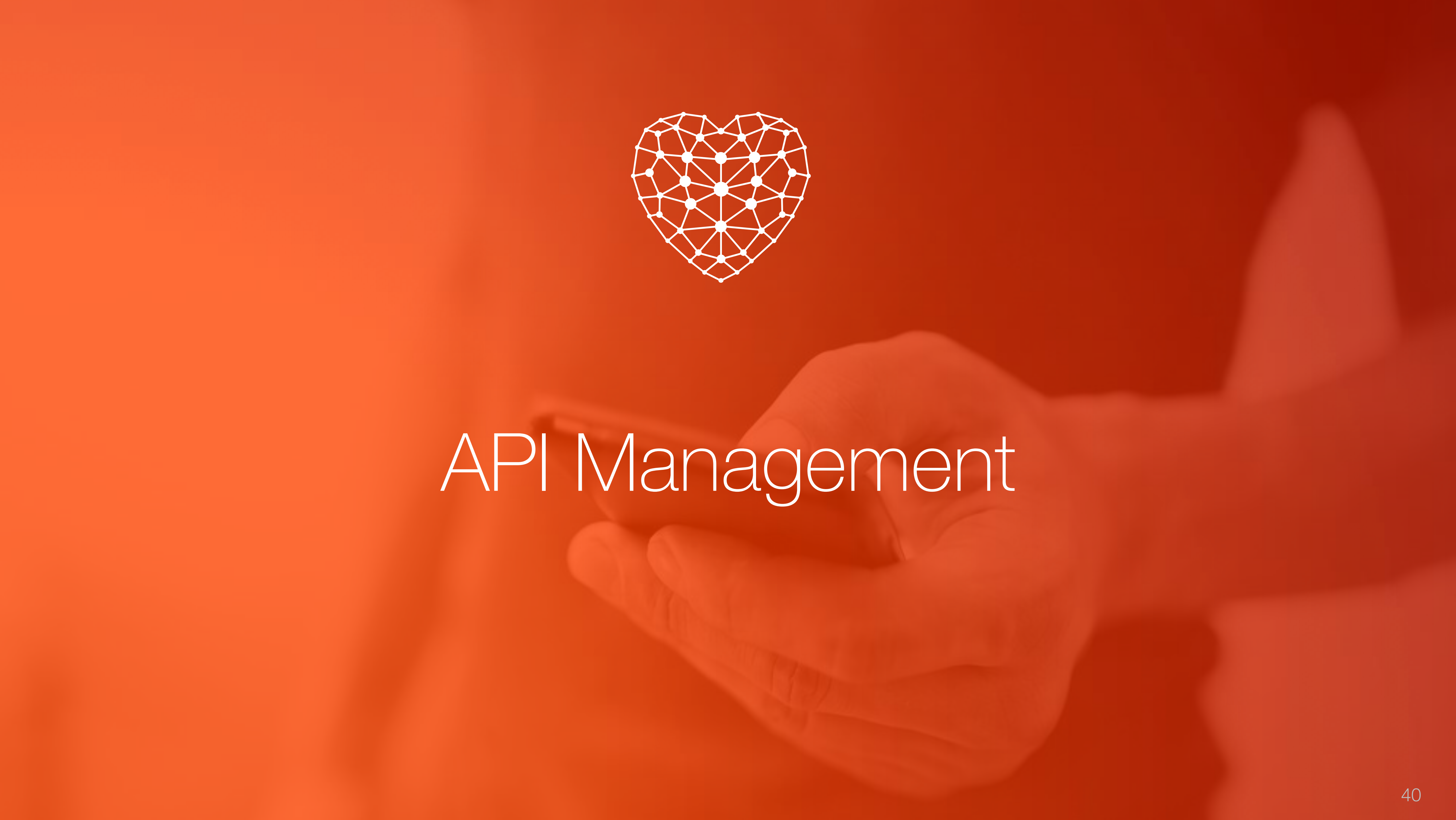
apigee

Exercise 1

Exercise to Create a proxy that connects to a basic authenticated backend (i.e. APIEatery Backend would work)

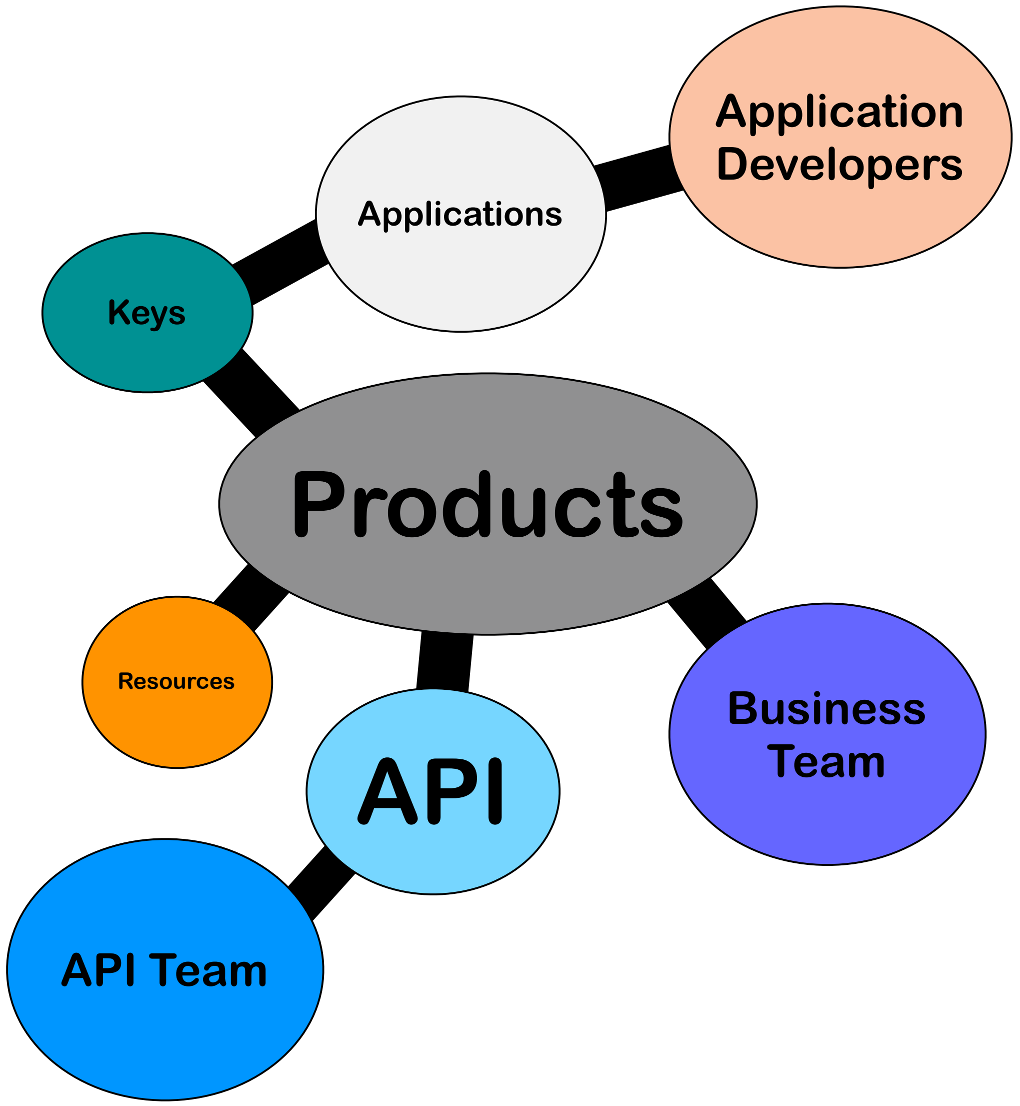
Step One: Create Proxy
Step Two: Create KVM
Step Three: Access KVM
Step Four: Build Authorization Header
Step Five: Test Server
Step Six (Optional): Create Target Servers
Step Seven (Optional): Configure Target Servers instead of the HTTP Target URL

apigee

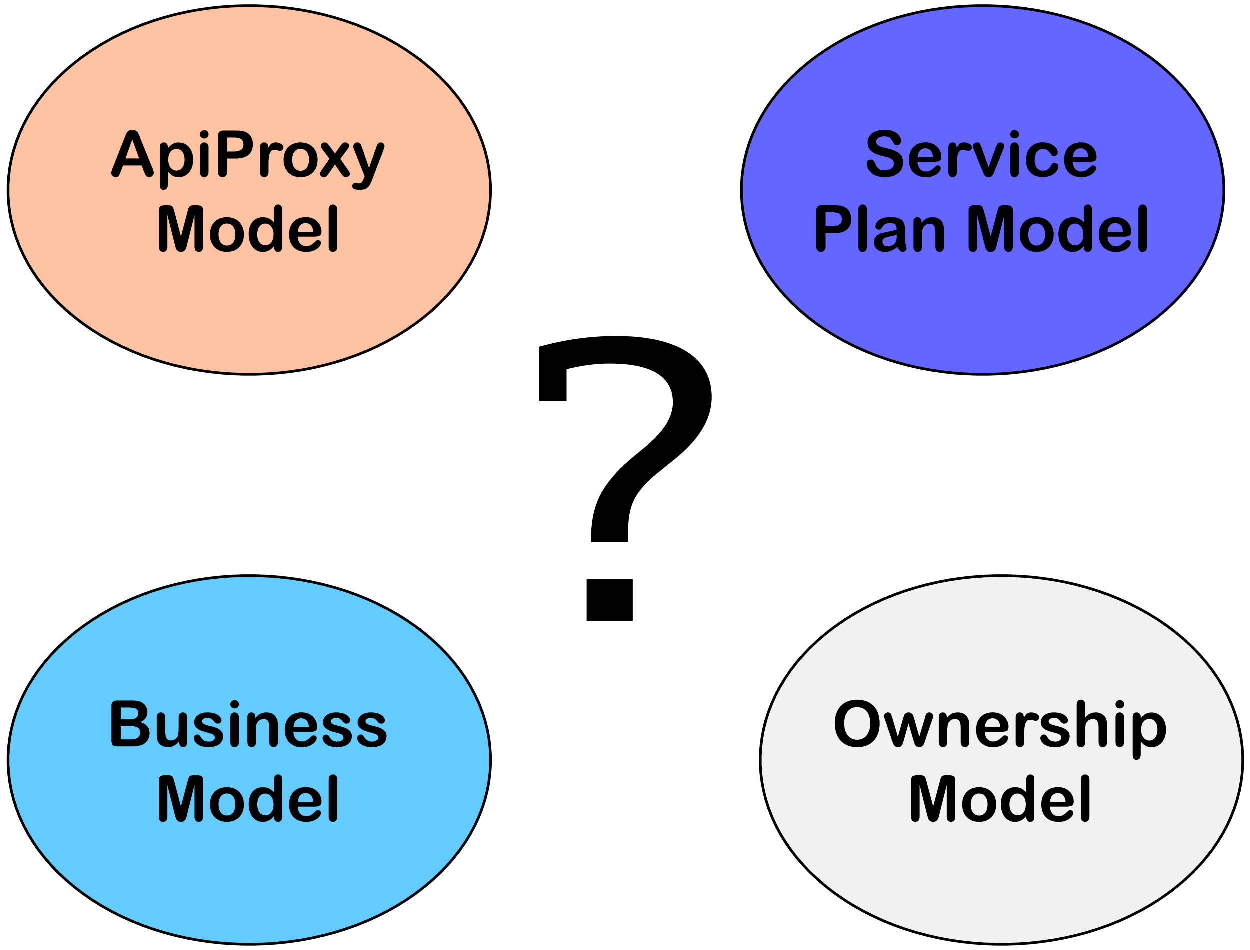


API Management

API Management Eco-System

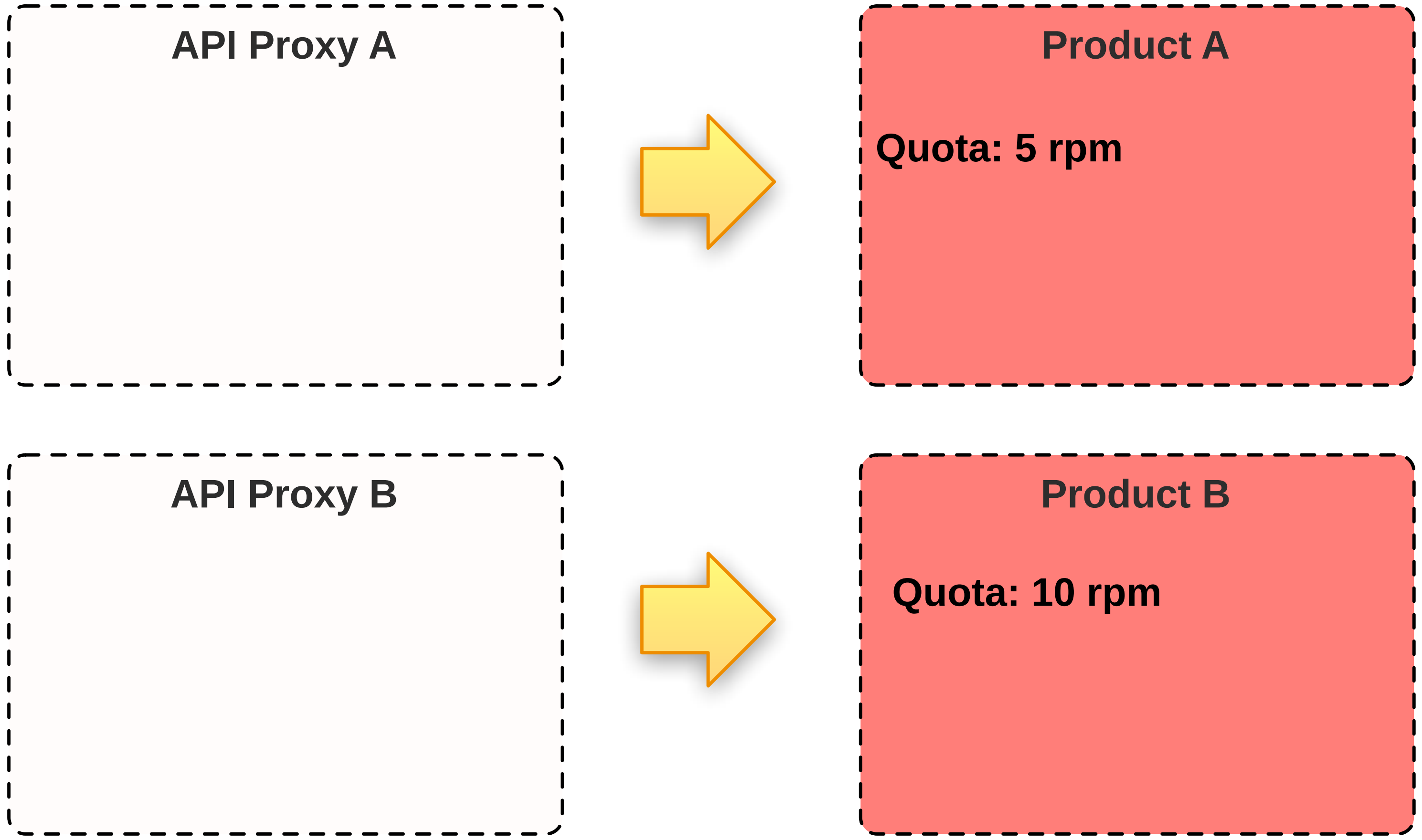


API Product Strategies



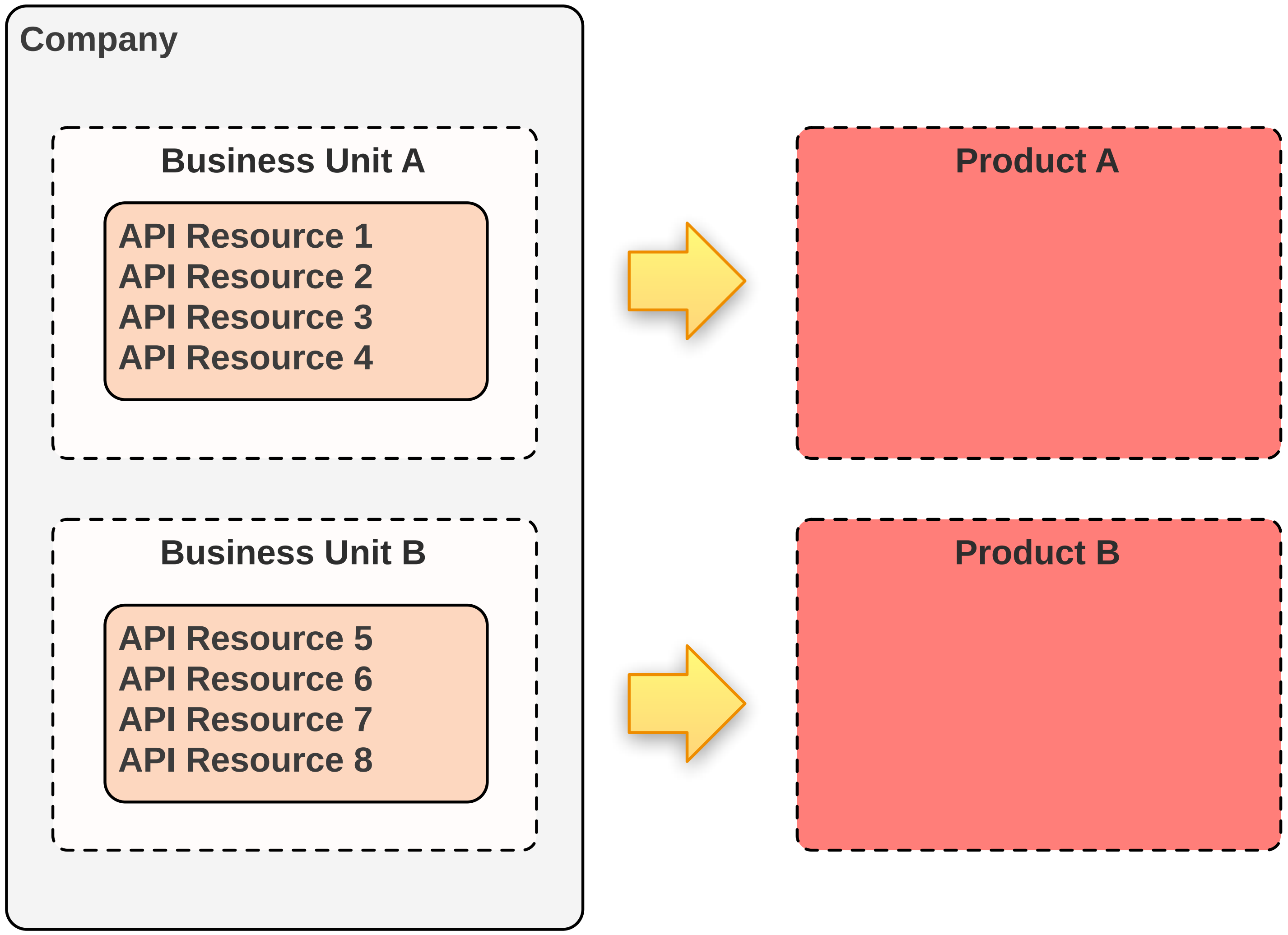
API Proxy Model

The API Proxy Model is a strategy centered around creating a 1 to 1 relationship with your API proxies and your products



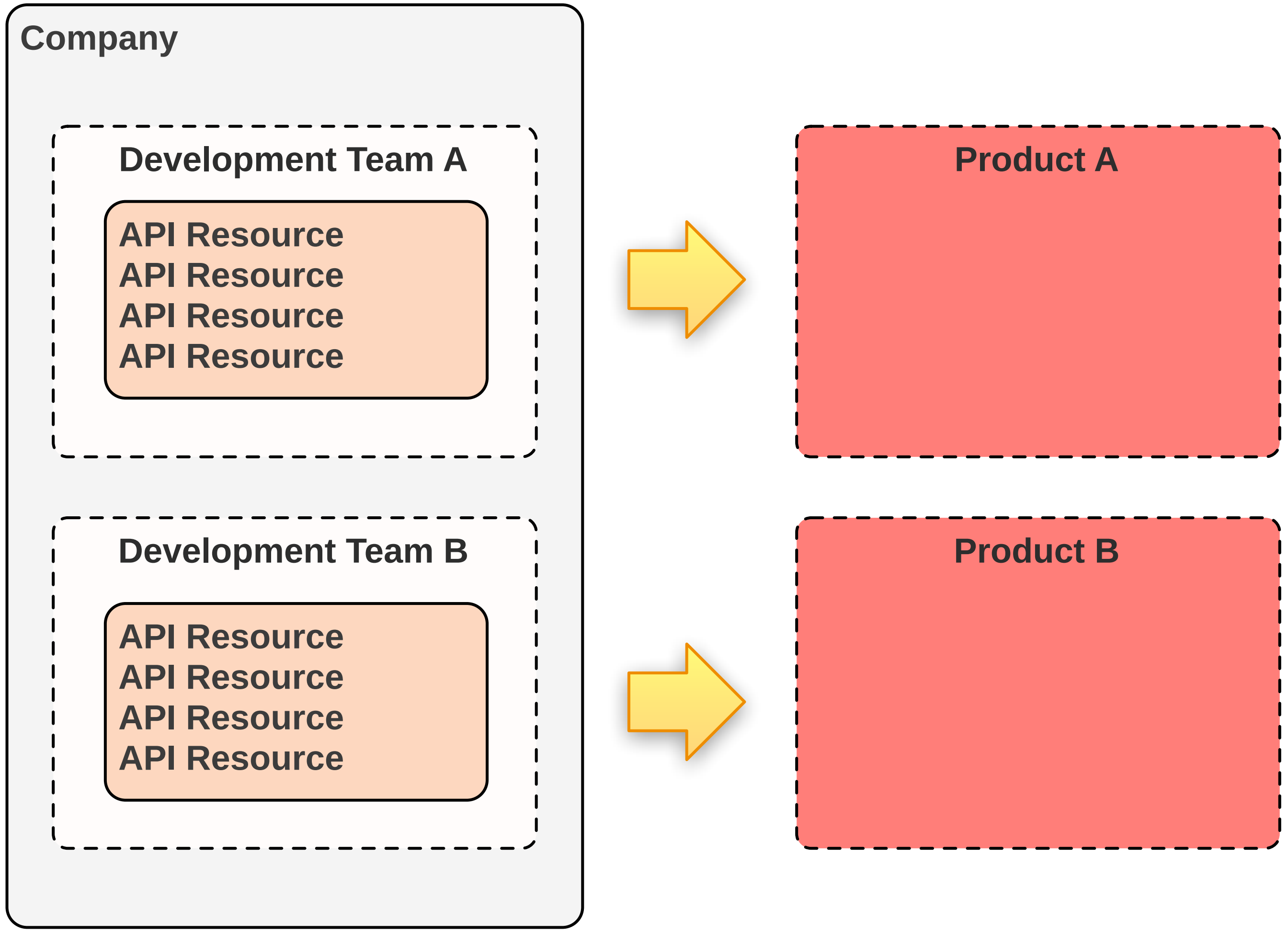
Business Model

The Business Model is a strategy centered around creating a relationship between a companies business unit specific API's and Products



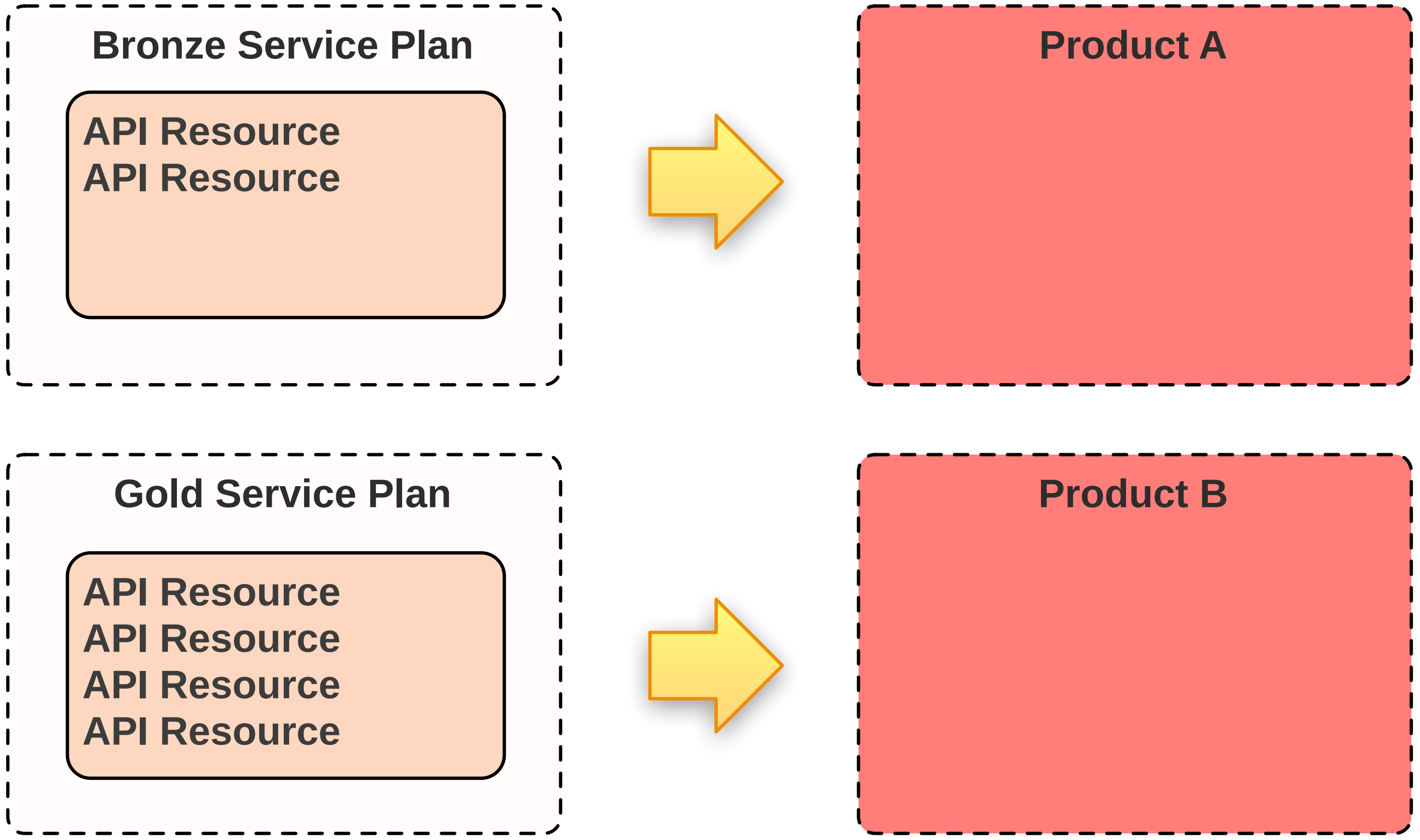
Ownership Model

The Ownership Model is a strategy that creates a relationship between the owner of the API resource and the Products



Service Plan Model

The Service Plan Model is Product strategy that groups API Resources in a Product based on a business model defined by different service levels e.g. Bronze, Silver, and Gold



Key Concepts

- Product to Application is a Many to Many Relationship
- When a Product is associated with an Application a Key is generated
- Products contain configuration used to display information an control workflow in the Developer Portal
- You can set custom configuration that can be accessed by every API that call that is made from an Application associated with a Product

How Products Manage Access Control

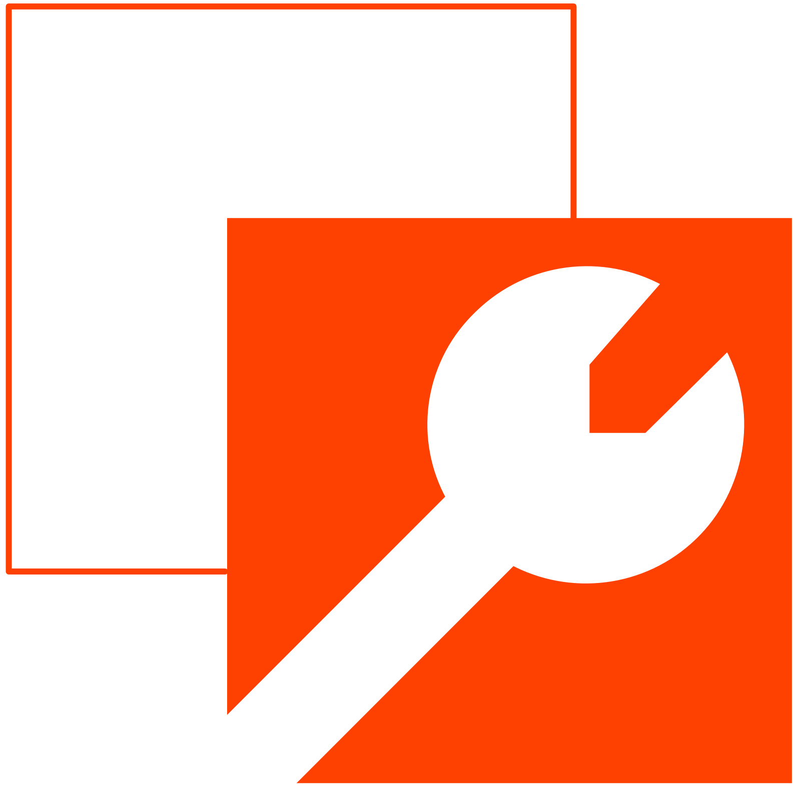
- By Default Access Control is can be restricted by either the combination of proxy and path-suffix or each individually
- Product resource validation happens when you validate either an Access Token or an API key
- Multiple product validations are done via a Union of the product definitions

apigee

How to extend API Products

- Products can be extended via custom attributes
- Variable flow.resource.name allows you to validate based on other information i.e. basepath, verb

Note: If your production environment lives in the same organization as your make sure you do an environment check, when validating your keys by using the flow.resource.name the system no longer checks the active environment



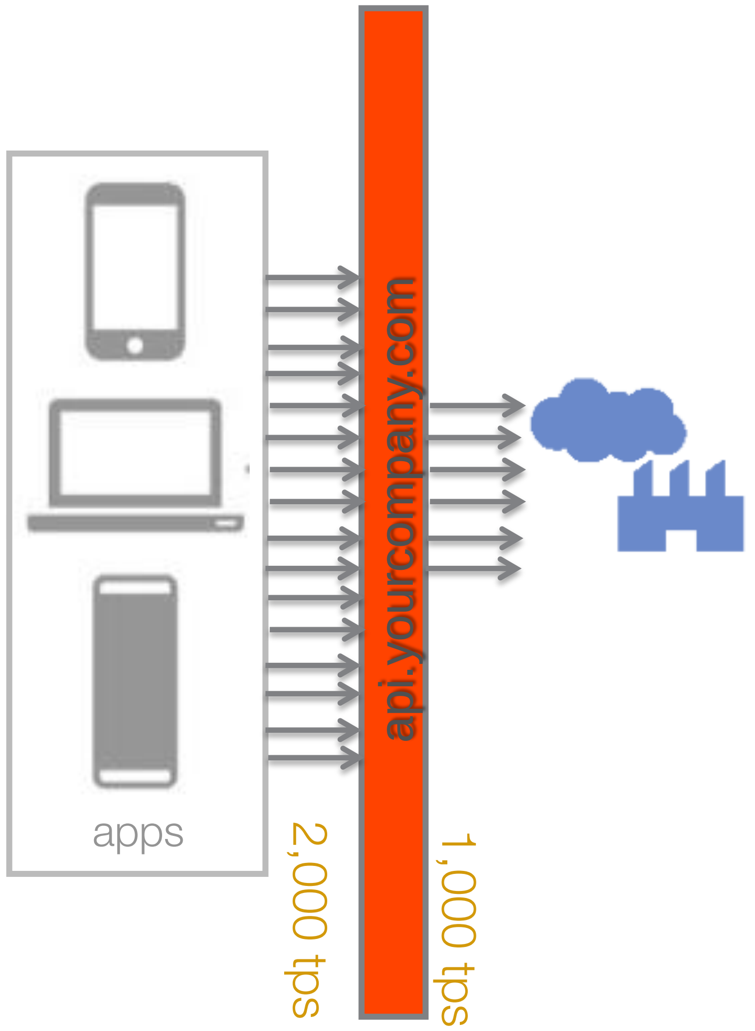
Demonstration

Rate Limiting: Spike Arrest

Spike Arrest

- SpikeArrest smooth's inflow request patterns over short intervals to ensure that demand does not outstrip capacity.

- SpikeArrest is a technical requirement to protect the backend as opposed to a Quota or Rate Limit which is a business requirement to manage developer relationships.



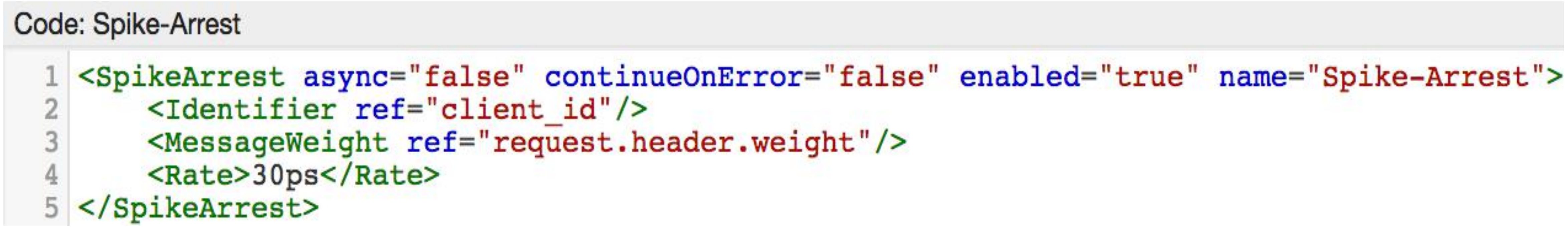
Spike Arrest and Denial of Service

- Spike Arrest Policy is often used offset the risk of DOS attack or a DDOS attack
- Denial of Service attacks is an attack on your platform instigated by sending thousands of requests against your API as a means to either bring down your platform or expose other vulnerabilities
- Spike Arrest monitors the rate at which traffic comes in, it does not count each request by putting them in timed buckets.

Spike Arrest Configuration

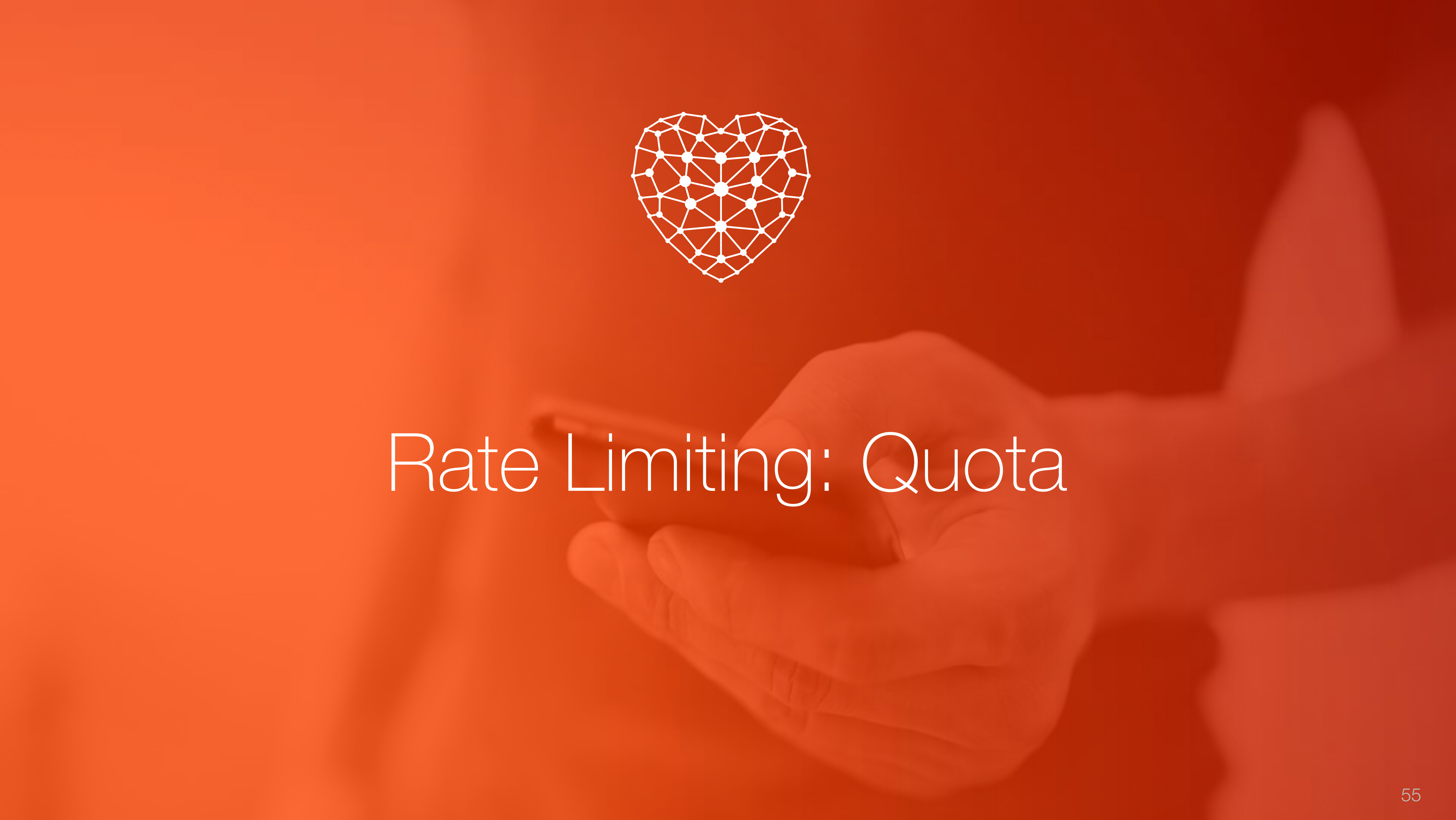
- Identifier
  - Mechanism to scope your incoming traffic
  - Uses a separate rate for each unique Id
- Message weight
  - Used to give extra weight to conditioned requests

- Rate
  - Specifies the rate you want to allow traffic to come into your API Proxy.
  - Rate is for each Message Processors
  - Rate is formatted with a number and a unit
  - Unit is defined as ps, pm (per second, per minute)



Code: Spike-Arrest

```xml
<SpikeArrest async="false" continueOnError="false" enabled="true" name="Spike-Arrest">
    <Identifier ref="client_id"/>
    <MessageWeight ref="request.header.weight"/>
    <Rate>30ps</Rate>
</SpikeArrest>
```



Rate Limiting: Quota

Quotas

- Use quota to set a defined amount of requests to an entity
- Quotas us buckets of request per time units unlike spike arrest which limits based of the rate of traffic
- Once a quota limit is reached all request generated by the limited entity will be rejected

Quota Configuration Part 1

- Identifier
  - Mechanism to scope your incoming traffic
  - Uses a separate bucket for each unique Id
- Message weight
  - Used to give extra weight to conditioned requests
- Allow
  - Specifies the number of requests allowed for a particular entity
- Time Unit
  - Month, Day, Week, Minute, Year
- Interval
  - Frequency of the time unit

Quota Type

Quota type indicates when the quota counter starts counting usage

- **calendar** (default if not specified)
  - Quota counting has an explicit start time
  - <StartTime> is specified and quota counting starts at the StartTime and is reset based on the specified <Interval> and <TimeUnit>
- **rollingwindow**
  - Quota uses a rolling window that resets based on the <Interval> and <TimeUnit>
  - The start time is the time the first message is received matching the <Identifier>
  - rollingwindow allows a quota that resets on every interval
- **flexi**
  - Start time is dynamic for each <Identifier> based on first message being received
  - Calls can be used until interval has elapsed
  - Quota does not automatically reset at the end of the interval
  - flexi allows access for a specified period of time

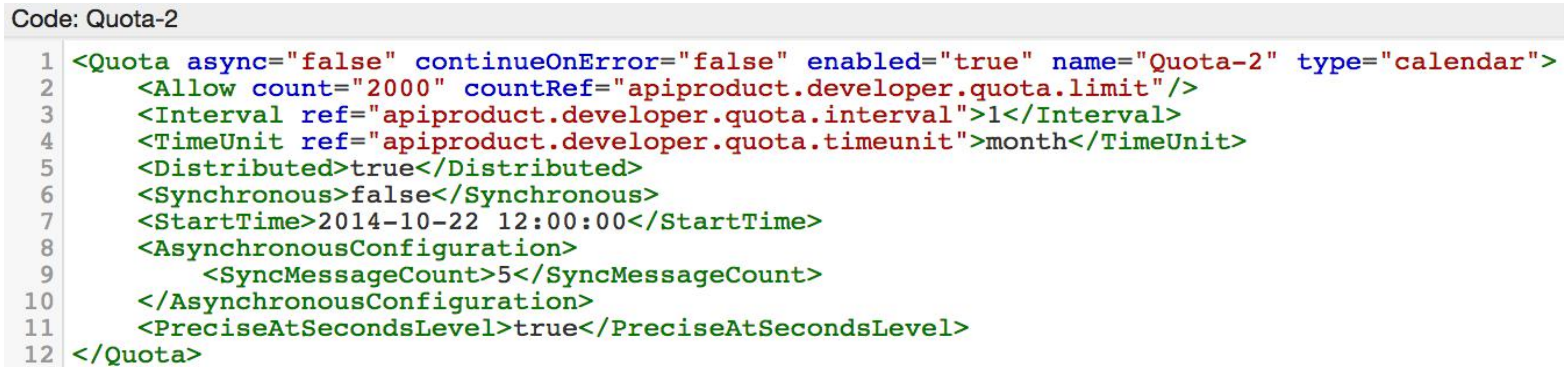
Quota type specified in **type** attribute of Quota root element

apigee

Distributed Quotas

- <Distributed> specifies whether the count is shared among all message processors (Distributed = true) or maintained separately for each message processor (Distributed = false)
- <Synchronous> specifies whether the distributed quota counter is updated synchronously
- <AsynchronousConfiguration> specifies how an asynchronous distributed quota is updated
- If <PreciseAtSecondsLevel> is set to true, the quota will be enforced with an accuracy of a second, even if the <TimeUnit> is set at a unit longer than a second

Example


Code: Quota-2

```
<Quota async="false" continueOnError="false" enabled="true" name="Quota-2" type="calendar">
    <Allow count="2000" countRef="apiproduct.developer.quota.limit"/>
    <Interval ref="apiproduct.developer.quota.interval">1</Interval>
    <TimeUnit ref="apiproduct.developer.quota.timeunit">month</TimeUnit>
    <Distributed>true</Distributed>
    <Synchronous>false</Synchronous>
    <StartTime>2014-10-22 12:00:00</StartTime>
    <AsynchronousConfiguration>
        <SyncMessageCount>5</SyncMessageCount>
    </AsynchronousConfiguration>
    <PreciseAtSecondsLevel>true</PreciseAtSecondsLevel>
</Quota>
```

Flow Variables

- Flow variables are created after a Quota policy executes
  - Variables are prefixed with "ratelimit.{policy_name}."
  - Examples are:

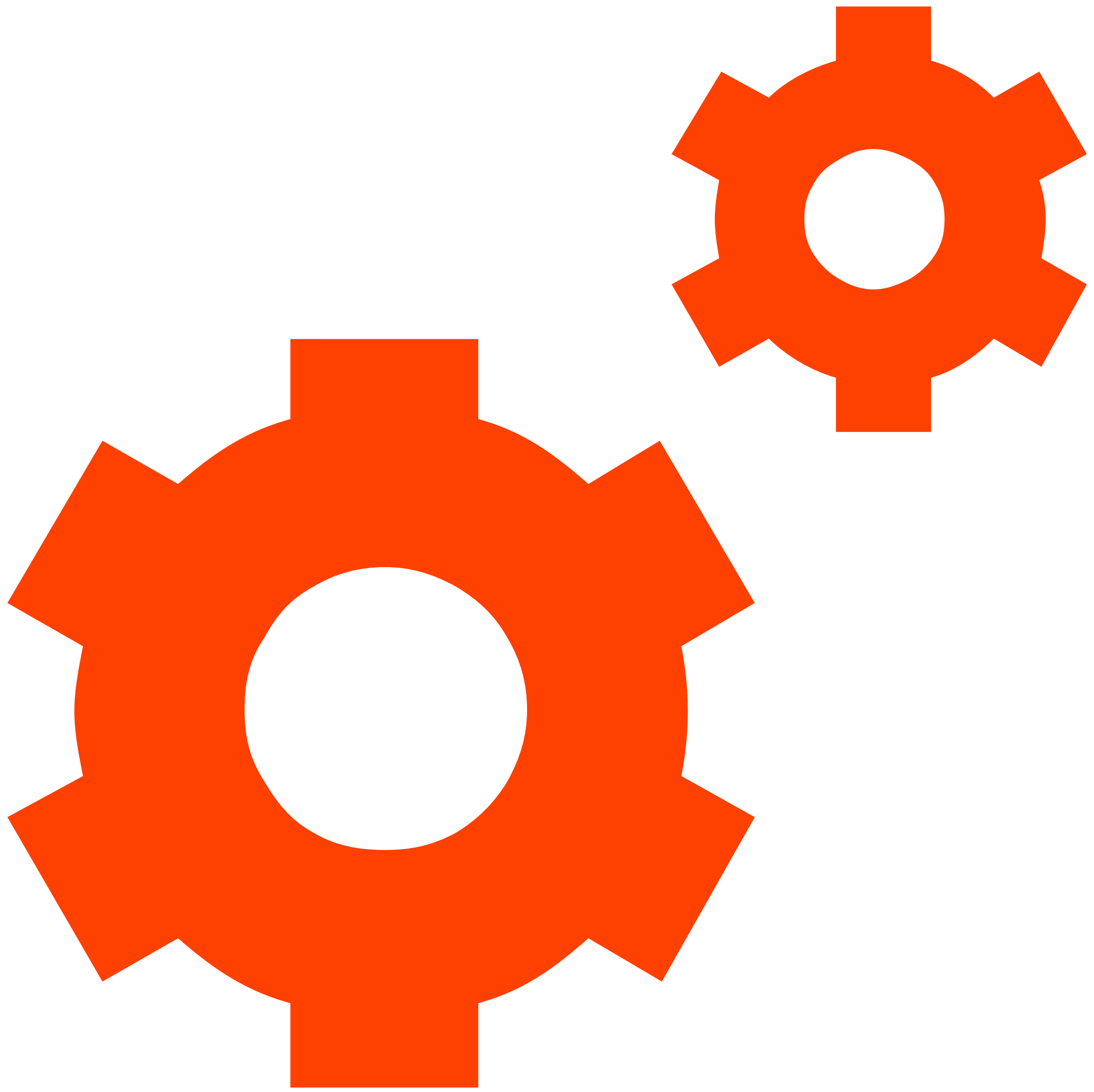
    ratelimit.{policy_name}.available.count (available quota count)

    ratelimit.{policy_name}.used.count (used quota count)

    ratelimit.{policy_name}.expiry.time (time in ms when quota resets)

    ratelimit.{policy_name}.identifier (identifier for the policy)

Setup Quota Via Product



apigee

Exercise

- Step One: Add Spike Arrest Policy
- Step Two: Add Quota Policy
- Step Three: Make Quota policy A-Sync
- Step Four: Feed Quota values from Product

Caching

Why Use Caching

Performance

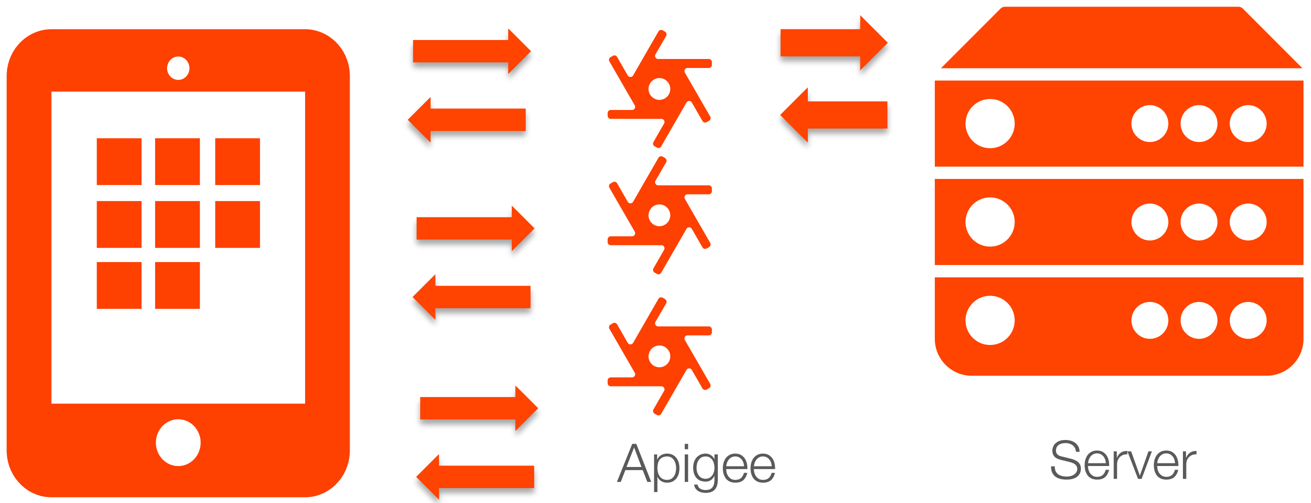
- Reducing Network latency
- Eliminate redundant request/response processing

Stability

- Reduce amount of load to backend services

Scalability

- Support higher TPS without adding additional hardware



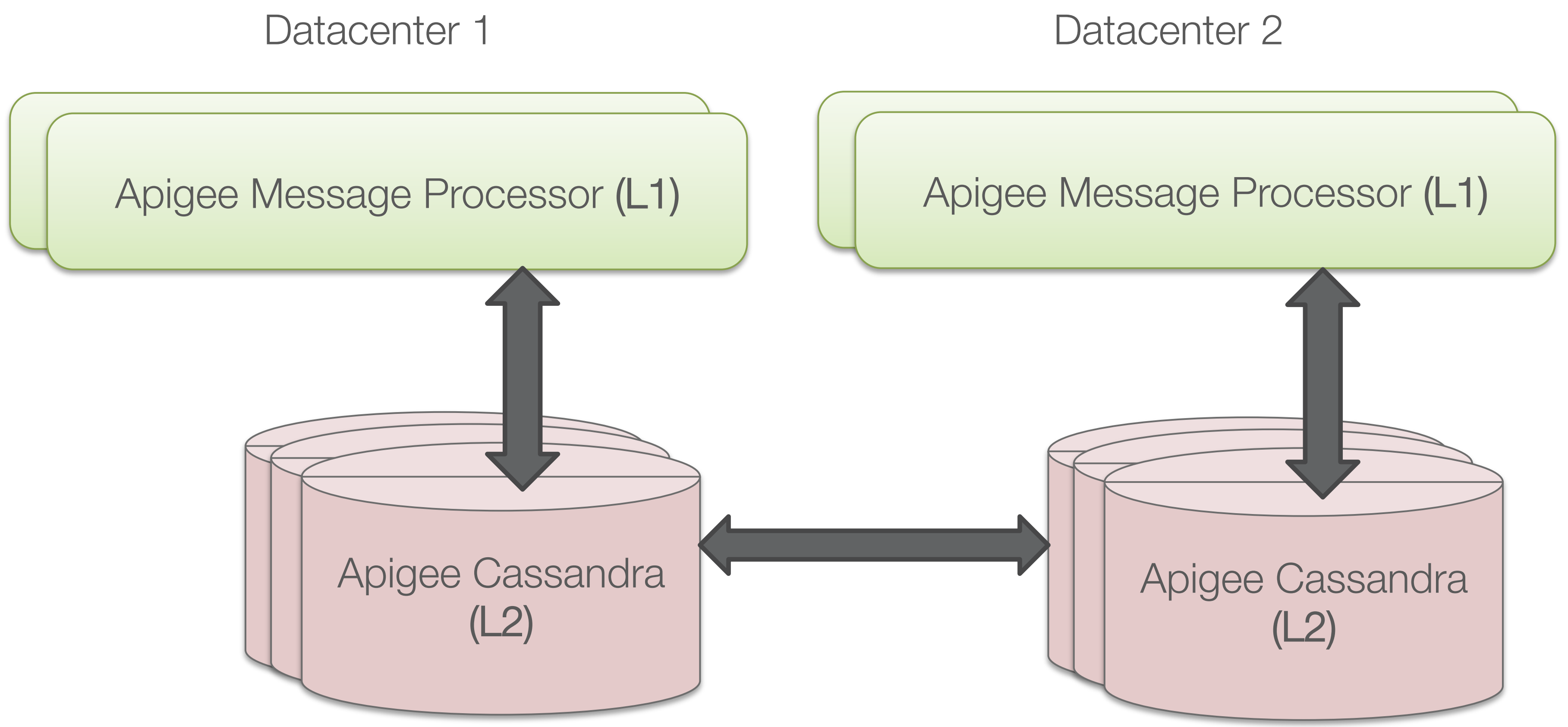
Caching Policies

- Response Cache
  - caches the entire HTTP response (headers, payload, etc.)
  - can set a different time-to-live for each entry
  - option for honoring HTTP cache headers for dynamic TTL (Expires, Cache-Control)
  - option to support caching of multiple formats based on request ‘Accept’ header

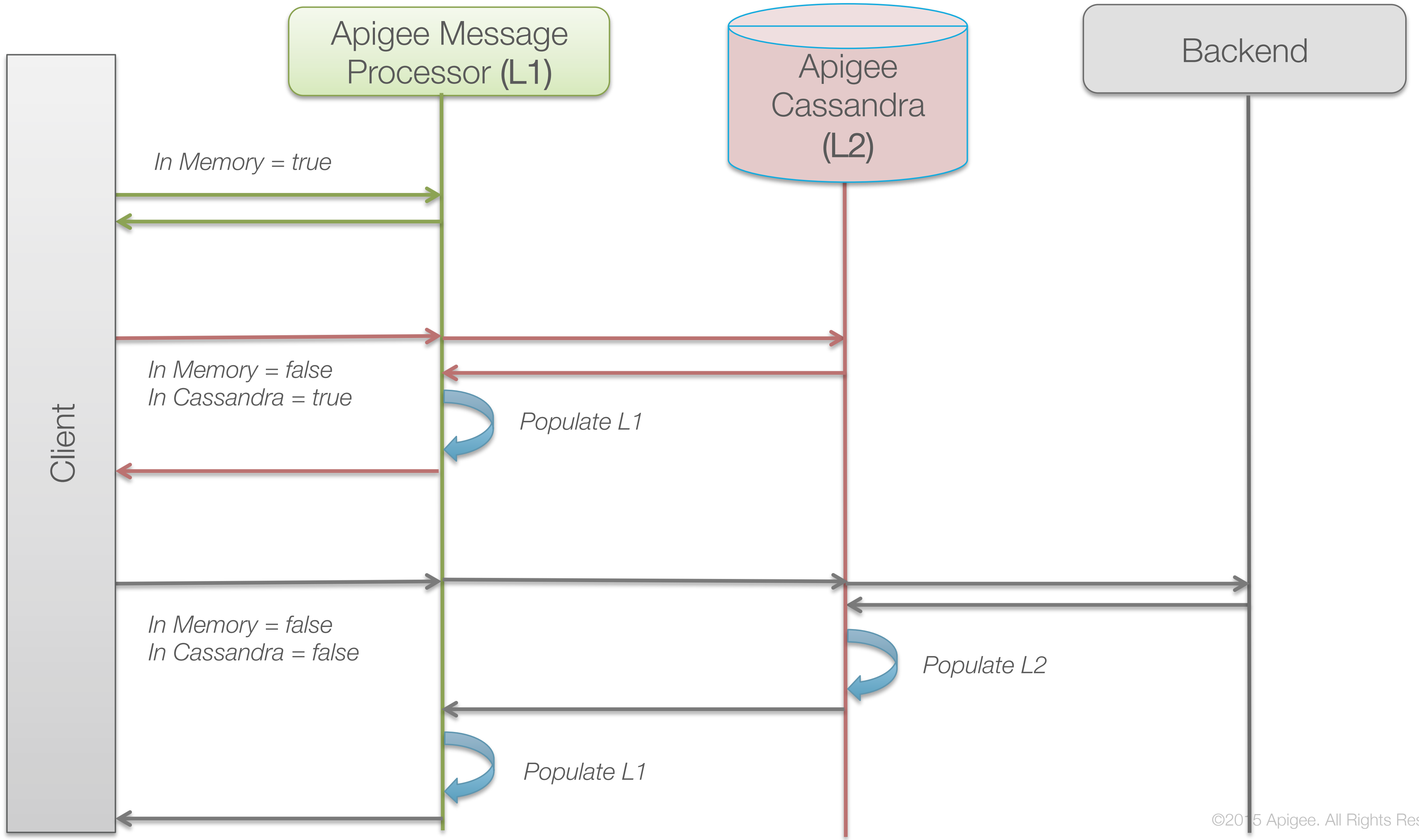
- Populate Cache/Lookup Cache
  - full control over caching, store any objects
  - Add/update and read entries using separate policies

Distributed Caching

Distributed caching allows cache entries to be distributed across multiple Message Processors within a region and across regions.



Cache Lookup/Populate Sequence



Response Cache

Reduces latency and network traffic by avoiding calls to backend and extra processing.

Cache Key

- can use multiple key fragments
- include a unique user identifier if user data is cached
- scoping affects the cache key

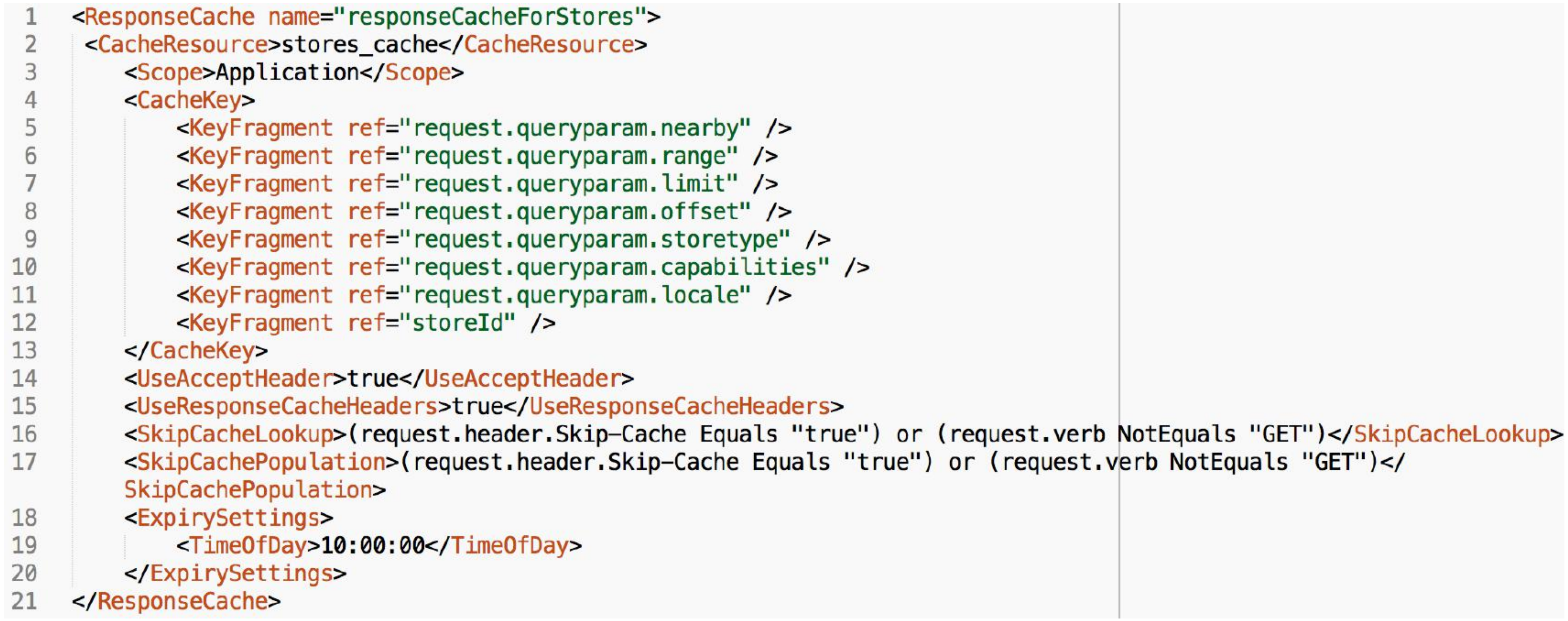
Expiration

- can specify a time of day or date the cache entry expires
- Leverage <UseResponseCacheHeaders> configuration so the policy dynamically sets TTL based on backend response HTTP cache headers.

Response Cache (cont'd)

- Optional conditions for skipping cache lookup or population.
- Same policy attached to request and response segments
  - When Request segment policy is reached, cache lookup happens
    - if cache hit, response is retrieved from cache and processing bypasses backend and other policies until ResponseCache policy in Response segment
    - if cache miss, request processing continues
  - When Response segment policy is reached, cache population happens and response message is stored

Response Cache (cont’d)



```
<ResponseCache name="responseCacheForStores">
 <CacheResource>stores_cache</CacheResource>
    <Scope>Application</Scope>
    <CacheKey>
        <KeyFragment ref="request.queryparam.nearby" />
        <KeyFragment ref="request.queryparam.range" />
        <KeyFragment ref="request.queryparam.limit" />
        <KeyFragment ref="request.queryparam.offset" />
        <KeyFragment ref="request.queryparam.storetype" />
        <KeyFragment ref="request.queryparam.capabilities" />
        <KeyFragment ref="request.queryparam.locale" />
        <KeyFragment ref="storeId" />
    </CacheKey>
    <UseAcceptHeader>true</UseAcceptHeader>
    <UseResponseCacheHeaders>true</UseResponseCacheHeaders>
    <SkipCacheLookup>(request.header.Skip-Cache Equals "true") or (request.verb NotEquals "GET")</SkipCacheLookup>
    <SkipCachePopulation>(request.header.Skip-Cache Equals "true") or (request.verb NotEquals "GET")</SkipCachePopulation>
    <ExpirySettings>
        <TimeOfDay>10:00:00</TimeOfDay>
    </ExpirySettings>
</ResponseCache>
```

Populate Cache / Lookup Cache

- Optimizes API performance by reducing request or response processing.
  - Can cache any object. Typically used after building custom data
  - Separate policies for cache population vs. lookup
  - Specify a time-to-live
  - Full control of population and lookup of cache via policies

Choosing which caching policy to use…

- Use Response Cache:
  - when identical requests and their corresponding identical responses are common (cache hits will be frequent)
  - to reduce unnecessary traffic to backend
  - to reduce latency for common requests
- Use Populate Cache / Lookup Cache:
  - when storing custom data objects (not always backend http response) to store data sets that have a specific maximum number of entries (vs. key/value maps)
  - to persist data across multiple API transactions

Cache Utilization and Optimization

The use of caching is only beneficial if requests are being served from cache. There are key things to ensure your cache solution is obtaining optimal performance.

– Ensure your cache key is built in such a way that the only request parameters used for the key are ones that dictate the client’s response. (e.g. don’t just use the URI as things like timestamps and other unique items can end up as part of the key resulting in 0% cache hits.

– Take advantage of the cache scope. Multiple APIs might utilize the same cache so set the scope to be ‘organization’ and re-use.

– Cache Performance dashboard is available in Apigee Analytics to report on cache hit % and performance gains.

Clearing the cache

Using the UI

– Allows you to clear the entries for a complete cache resource
– Typically used to invalidate all cache entries

Management API

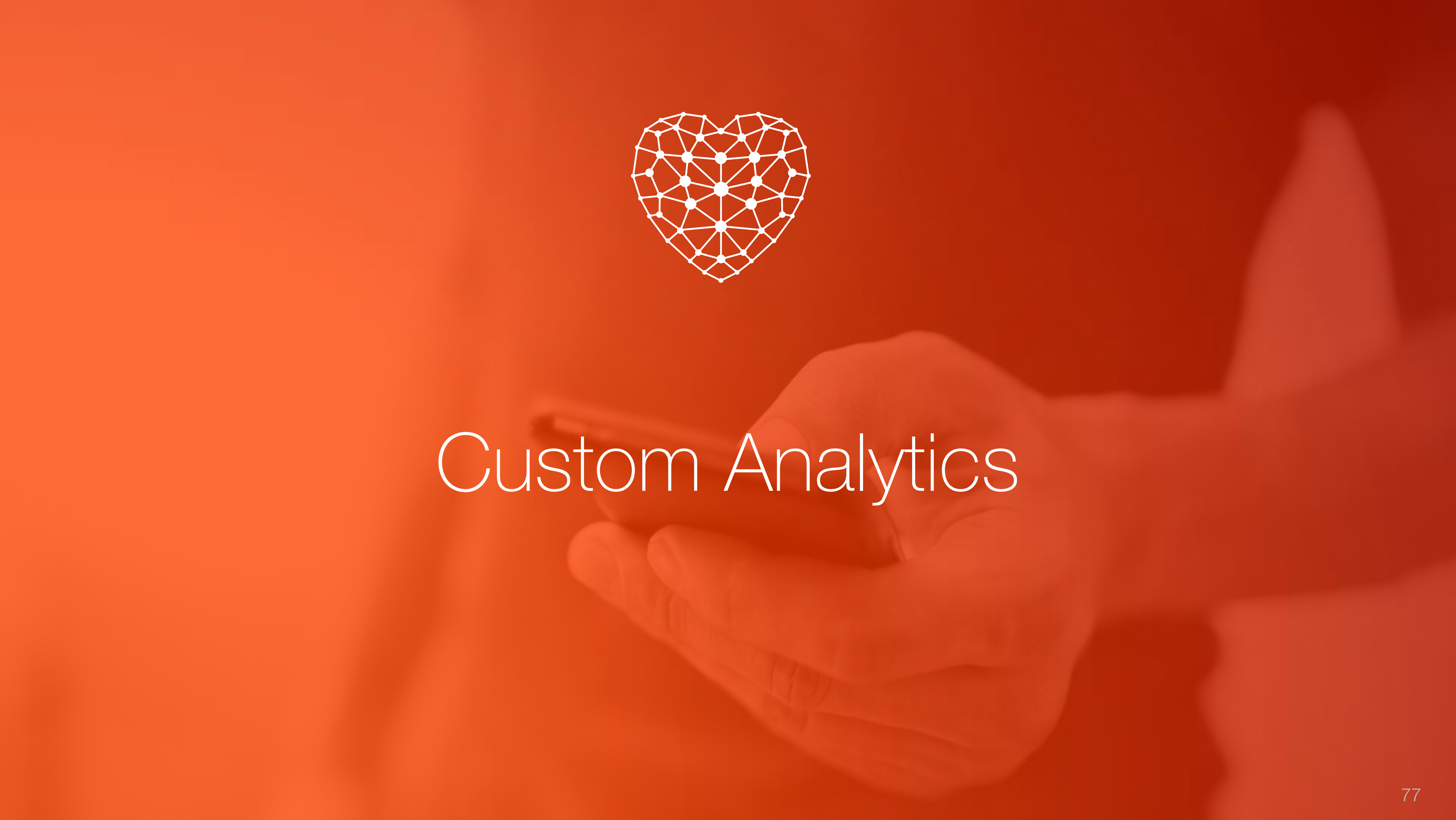
– Same as UI with the addition of being able to specify a prefix used for cache keys
– Typically used to invalidate all cache entries

Invalidate Cache Policy

– Allows you build a fully customized API resource to invalidate specific entries
– Used in implementations to invalidate specific cache entries

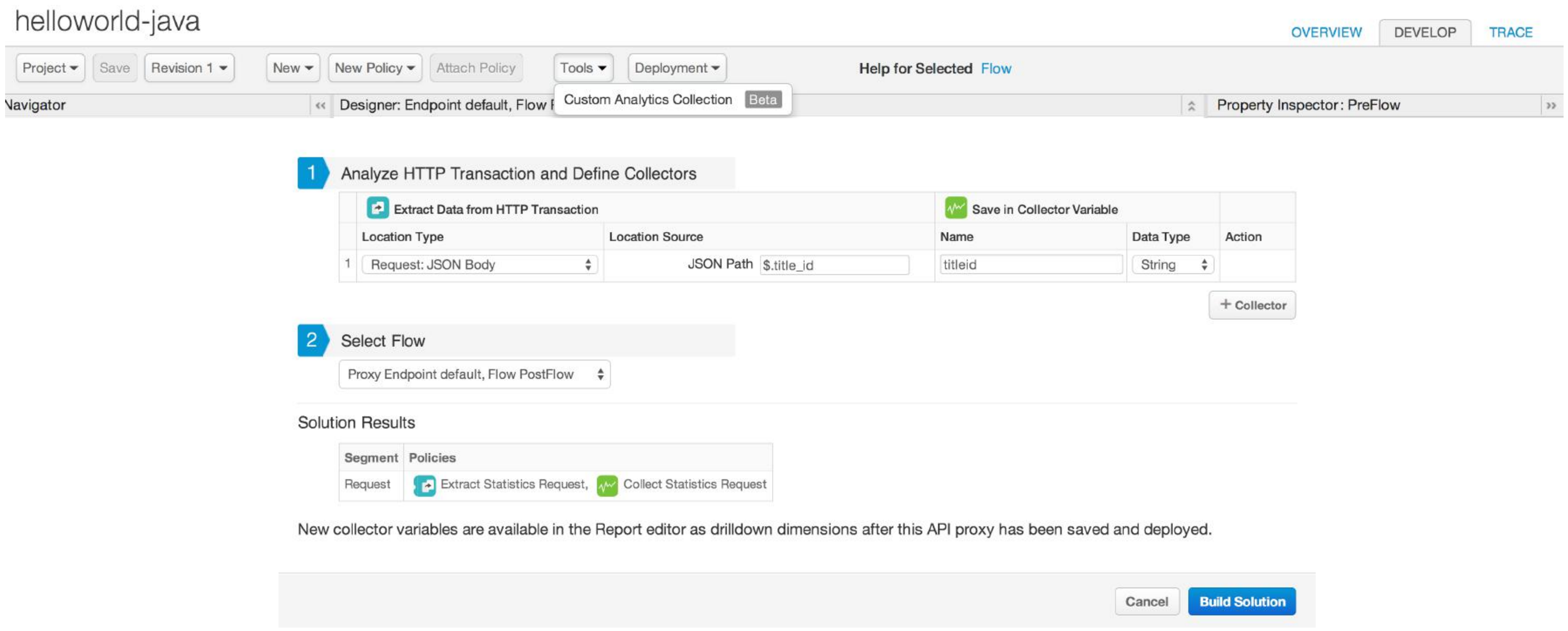
Exercise

- Step One: Add and Configure response cache policy
- Step Two: Add and Configure Populate and Lookup Policies



Custom Analytics

Custom Statistics Collection



helloworld-java

1 Analyze HTTP Transaction and Define Collectors

	Extract Data from HTTP Transaction		Save in Collector Variable		
	Location Type	Location Source	Name	Data Type	Action
1	Request: JSON Body	JSON Path $.title_id	titleid	String	

2 Select Flow

Proxy Endpoint default, Flow PostFlow

Solution Results

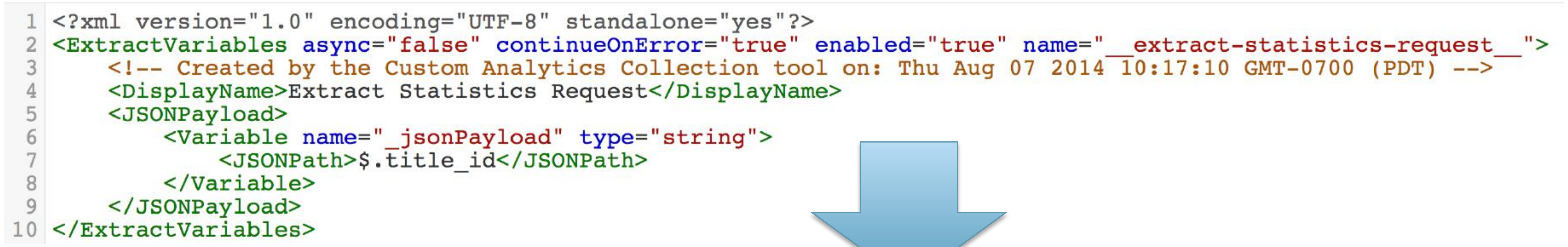
Segment	Policies
Request	Extract Statistics Request, Collect Statistics Request

New collector variables are available in the Report editor as drilldown dimensions after this API proxy has been saved and deployed.

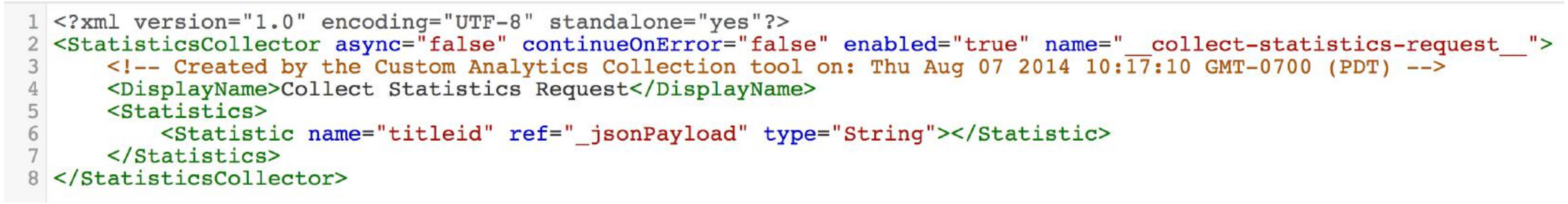
Apigee allows developers to send any custom data in its out-of-box Analytics engine. This is available through a policy, and even a wizard to set up for you.

Custom Statistics Collection

- Policies are auto-generated by the wizard
- Can be manually customized to extract any part of the HTTP message for used in the StatisticsCollector policy

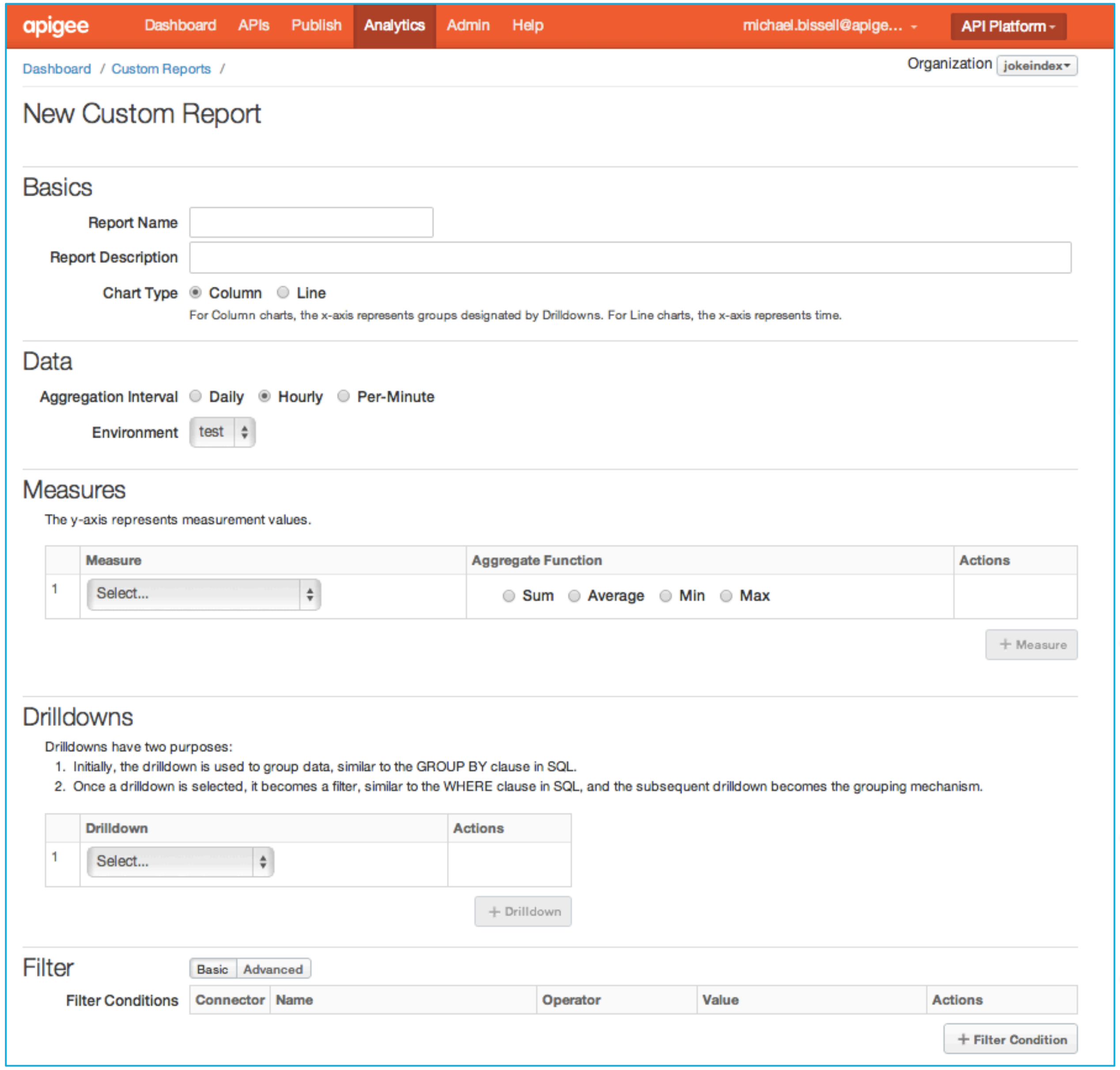


```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<ExtractVariables async="false" continueOnError="true" enabled="true" name="__extract-statistics-request__">
    <!-- Created by the Custom Analytics Collection tool on: Thu Aug 07 2014 10:17:10 GMT-0700 (PDT) -->
    <DisplayName>Extract Statistics Request</DisplayName>
    <JSONPayload>
        <Variable name="_jsonPayload" type="string">
            <JSONPath>$.title_id</JSONPath>
        </Variable>
    </JSONPayload>
</ExtractVariables>
```



```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<StatisticsCollector async="false" continueOnError="false" enabled="true" name="__collect-statistics-request__">
    <!-- Created by the Custom Analytics Collection tool on: Thu Aug 07 2014 10:17:10 GMT-0700 (PDT) -->
    <DisplayName>Collect Statistics Request</DisplayName>
    <Statistics>
        <Statistic name="titleid" ref="_jsonPayload" type="String"></Statistic>
    </Statistics>
</StatisticsCollector>
```

Analytics: Custom Reporting via UI



New Custom Report

Basics

Report Name	
Report Description	
Chart Type	[✓ Column] Line

For Column charts, the x-axis represents groups designated by Drilldowns. For Line charts, the x-axis represents time.

Data

Aggregation Interval	Daily [✓ Hourly] Per-Minute
Environment	test

Measures

The y-axis represents measurement values.

	Measure	Aggregate Function	Actions
1	Select...	Sum Average Min Max	

Drilldowns

Drilldowns have two purposes:

1. Initially, the drilldown is used to group data, similar to the GROUP BY clause in SQL.
2. Once a drilldown is selected, it becomes a filter, similar to the WHERE clause in SQL, and the subsequent drilldown becomes the grouping mechanism.

	Drilldown	Actions
1	Select...	

Filter

Filter Conditions

Connector	Name	Operator	Value	Actions

← Report Name and Type

← Format and Environment

← Primary Reporting Measures

← Drilldown Reporting Measures

← Additional Filters

apigee