Apigee Edge Training

Anatomy of an API proxy

Digging into our new Proxy

- Endpoints
- Virtual Hosts
- Endpoint Properties
- Flows
- Polices

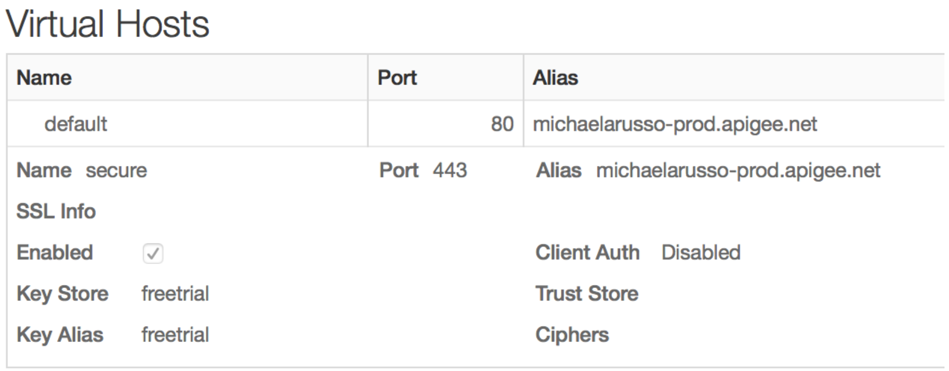
API Proxy and Target Endpoints

- Define an API as a series of resources that access a given target system.
- Client-side interfaces can be access using either HTTP or HTTPS
- Targets can be accessed using either HTTP or HTTPS, using either one-way SSL or two-way SSL with mutual authentication
- REST and SOAP targets supported
- “First match” selection: define a set of resource criteria matching incoming requests, and the first match found controls request execution
- Resource matching on path nodes, query parameters, HTTP verbs and other types of conditions
- Determine target service for requests using either static or dynamic routing

Virtual Hosts

A environment can contain multiple virtual hosts and a proxy's client side endpoint is configured to listen on one or more virtual hosts. The virtual host's configuration determines how an API request will route into an API proxy

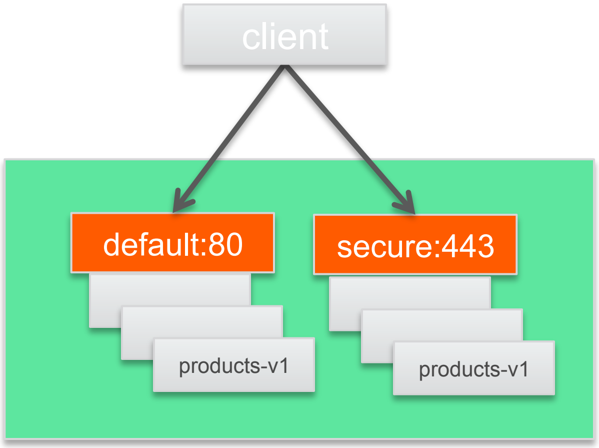
Environment Configuration



Virtual Hosts

Name	Port	Alias
default	80	michaelarusso-prod.apigee.net

Name secure	Port 443	Alias michaelarusso-prod.apigee.net	
SSL Info			
Enabled	✓	Client Auth	Disabled
Key Store	freetrial	Trust Store	
Key Alias	freetrial	Ciphers	



Apigee classifies requests using the following:

- Virtual Host Port
- Request's Host Alias (value of Host header)
- Proxy Endpoint BasePath

```
<HTTPProxyConnection>
    <BasePath>/v1/products</BasePath>
    <VirtualHost>default</VirtualHost>
    <VirtualHost>secure</VirtualHost>
</HTTPProxyConnection>
```

Endpoint Properties

In addition to the standard endpoint configurations there are some commonly used 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

<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.

Proxy Endpoint Route Rules

- Target is determined by RouteRules
  - One or more RouteRule sections at the bottom of the proxy definition file (proxies/{proxy}.xml) within ProxyEndpoint

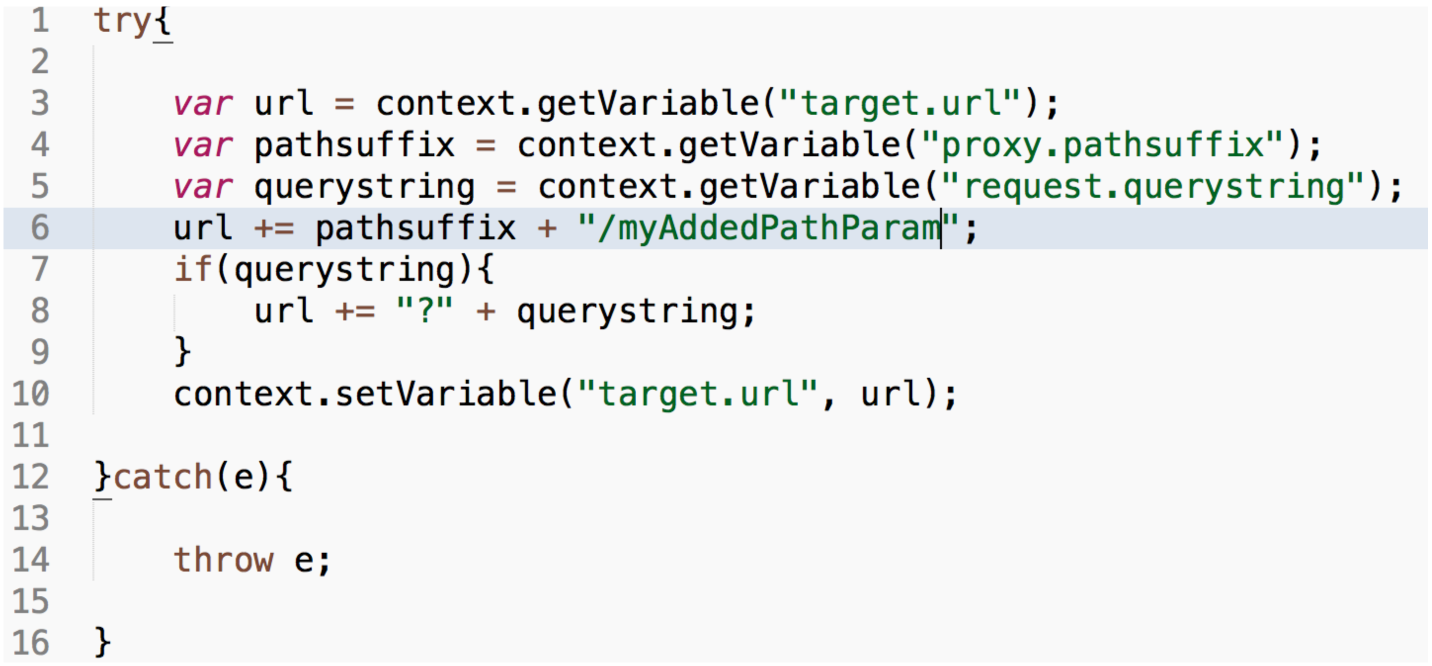
```
<RouteRule name="auth">
        <Condition>proxy.pathsuffix MatchesPath "/auth"</Condition>
        <!-- no target called -->
</RouteRule>
<RouteRule name="routeToTestServer">
        <Condition>request.header.X-TestServer == "true"</Condition>
        <TargetEndpoint>testServer</TargetEndpoint>
</RouteRule>
<RouteRule name="default">
        <!-- no condition, so always executes if it reaches this far -->
        <TargetEndpoint>default</TargetEndpoint>
</RouteRule>
```

  - RouteRules evaluated top to bottom, taking first match only
  - TargetEndpoint is the name of the target file
    - No TargetEndpoint means no target called (no Target Request and Target Response)
  - No condition supplied means that RouteRule will automatically match

Advanced Routing

Once the Route Rules have selected a TargetEndpoint and processing has began inside the Target, you can then apply logic to route the request to specific paths in the backend.

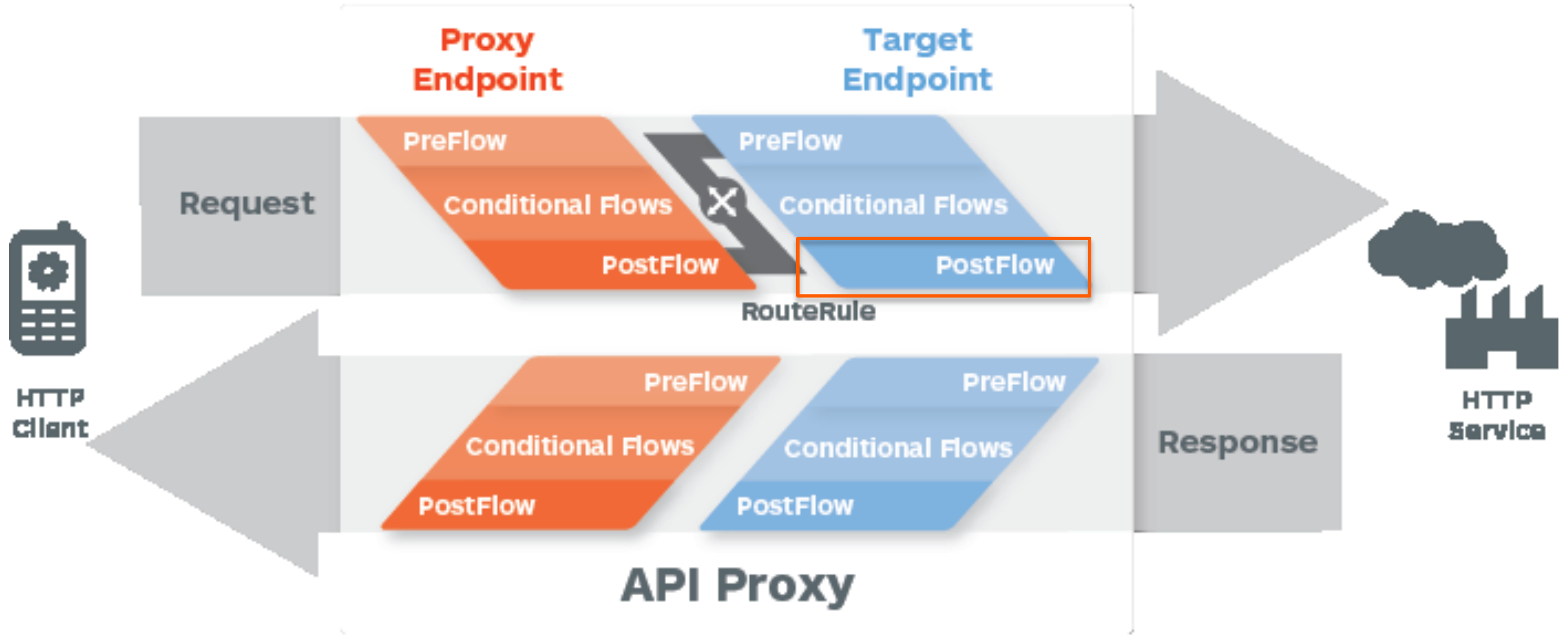
- ***target.url*** variable for read/write. You can override this via policy, but when doing so it stops the automatic copying of the request querystring.



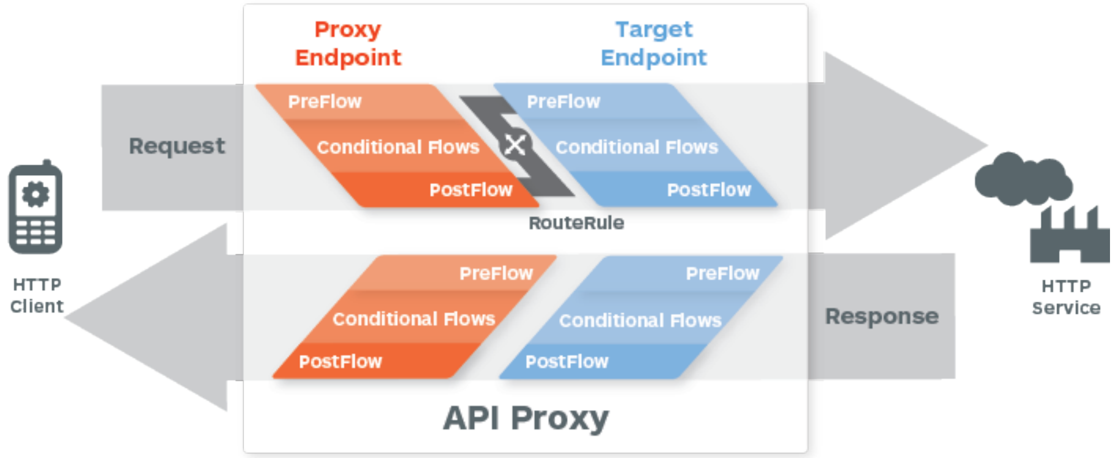
```
try{

    var url = context.getVariable("target.url");
    var pathsuffix = context.getVariable("proxy.pathsuffix");
    var querystring = context.getVariable("request.querystring");
    url += pathsuffix + "/myAddedPathParam";
    if(querystring){
        url += "?" + querystring;
    }
    context.setVariable("target.url", url);

}catch(e){

    throw e;

}
```

Policy Flows

"Policies" are steps within the API Flow. The Proxy Endpoint contains policies that affect the request and response to the Client. The Target Endpoint contains policies that affect the request and response to the backend Target server



API Flow Processing



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>
  ...
```

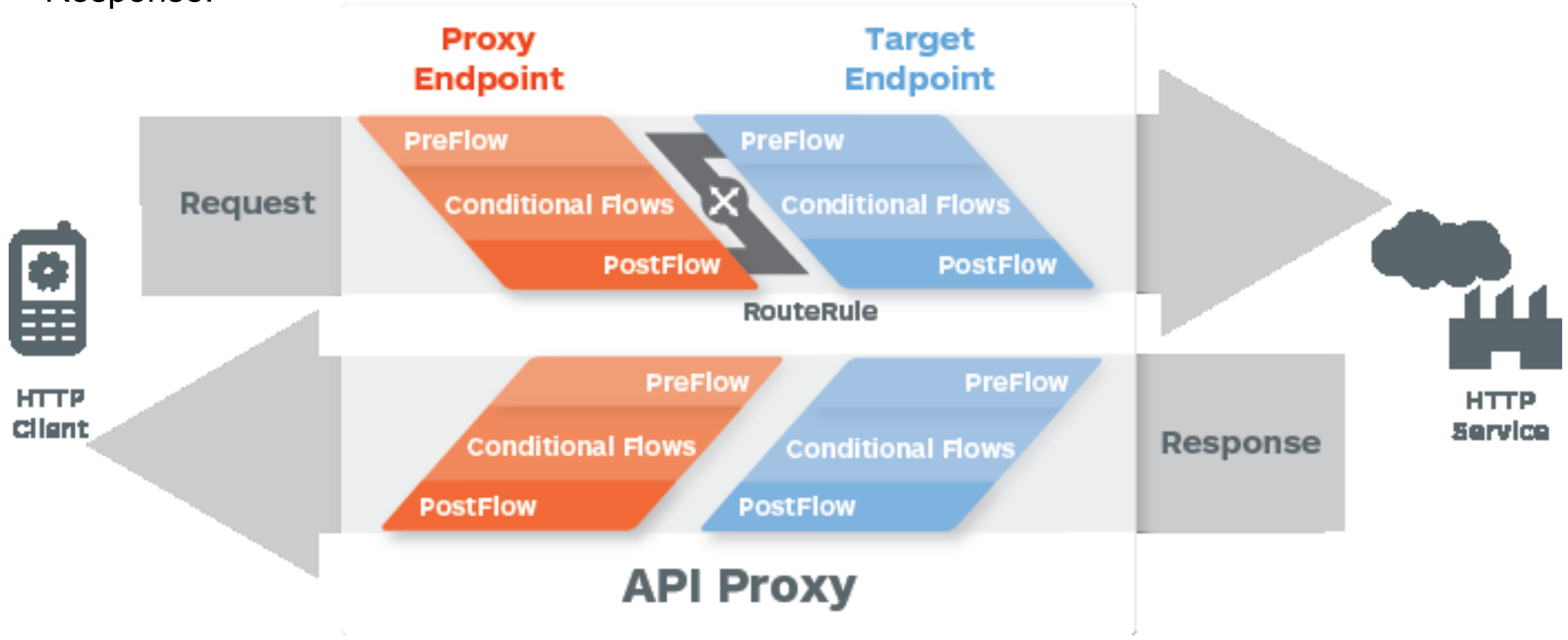
Post Response Flow

- PostClientResponseFlow
  - Lives in the proxy definition
  - Can only have Message Logging policies attached
  - Used as a method to alleviate the latency associated with logging to a syslog server.

Adding Resources

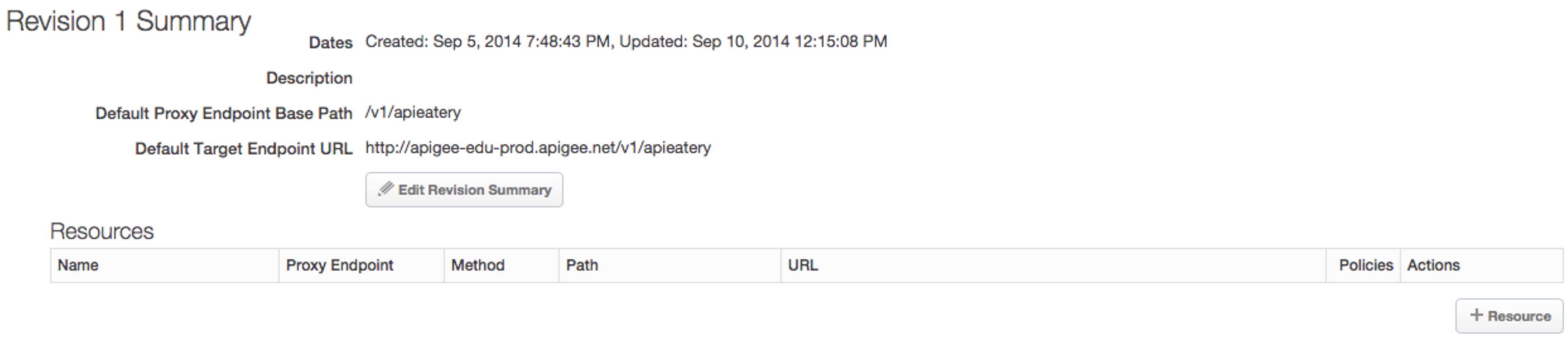
Conditional Flows / Resources

A "Conditional Flow" is a collection of policies within the API Flow that all share one rule. All policy attachments in the Preflow execute first, then the first matching Conditional Flow execute, then everything in the Postflow executes. This happens on the Proxy Request, then the Target Request, and repeats back through the Target Response and the Proxy Response.



```
<Condition>(proxy.pathsuffix MatchesPath &quot;/trucks&quot;) and (request.verb = &quot;GET&quot;)</Condition>
```

Adding Resources (Conditional Flows)

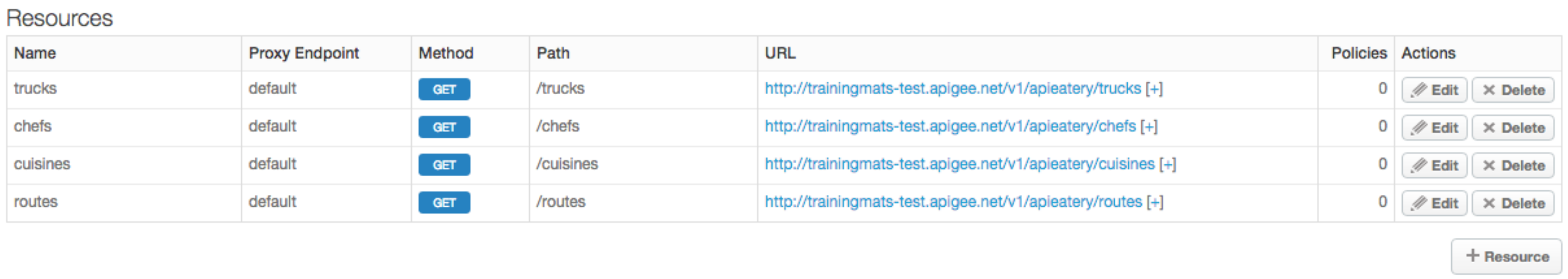


Revision 1 Summary

Dates	Created: Sep 5, 2014 7:48:43 PM, Updated: Sep 10, 2014 12:15:08 PM
Description	
Default Proxy Endpoint Base Path	/v1/apieatery
Default Target Endpoint URL	http://apigee-edu-prod.apigee.net/v1/apieatery

Resources

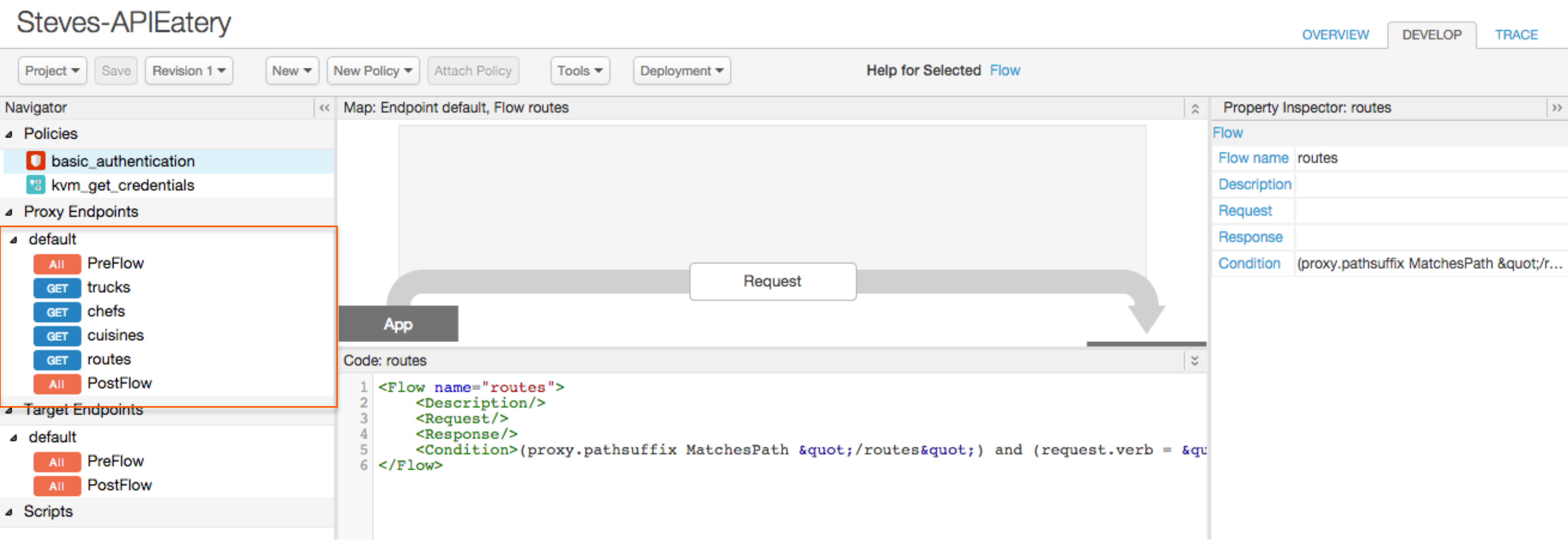
Name	Proxy Endpoint	Method	Path	URL	Policies	Actions



Resources

Name	Proxy Endpoint	Method	Path	URL	Policies	Actions
trucks	default	GET	/trucks	http://trainingmats-test.apigee.net/v1/apieatery/trucks [+]	0	Edit ✕ Delete
chefs	default	GET	/chefs	http://trainingmats-test.apigee.net/v1/apieatery/chefs [+]	0	Edit ✕ Delete
cuisines	default	GET	/cuisines	http://trainingmats-test.apigee.net/v1/apieatery/cuisines [+]	0	Edit ✕ Delete
routes	default	GET	/routes	http://trainingmats-test.apigee.net/v1/apieatery/routes [+]	0	Edit ✕ Delete

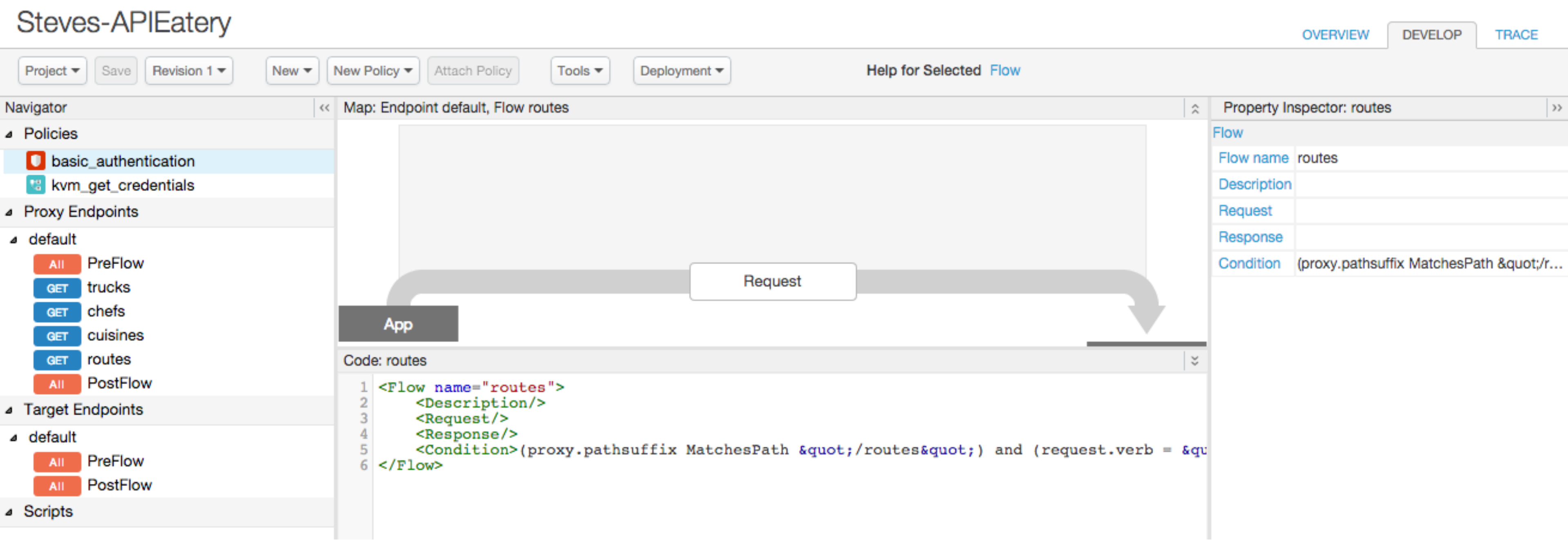
Resource Menu



Flow name	routes
Description	
Request	
Response	
Condition	(proxy.pathsuffix MatchesPath &quot;/r...

```
<Flow name="routes">
    <Description/>
    <Request/>
    <Response/>
    <Condition>(proxy.pathsuffix MatchesPath &quot;/routes&quot;) and (request.verb = &qu
</Flow>
```

Conditional Flows in XML



Flow	
Flow name	routes
Description	
Request	
Response	
Condition	(proxy.pathsuffix MatchesPath &quot;/r...

```
<Flow name="routes">
    <Description/>
    <Request/>
    <Response/>
    <Condition>(proxy.pathsuffix MatchesPath &quot;/routes&quot;) and (request.verb = &qu
</Flow>
```

Revisit Lab1

- We created pass through proxy in lab1 for ratings API
  - Is it the right way to setup a proxy?
  - What are the limitations of setting it this way?
  - How can we fix this?

Day 1 – Lab 2

- Adjust the proxy created in Lab1 to set it up the right way
  - Create a new resource as follows
    - Name = rating
    - Method = GET
    - Path = /ratings
  - Update the target URL and remove /ratings
- Test your proxy using Postman

Day 1 – Lab 3 – Dynamic Routing

- Create a new resource as follows:
  - Name=chefs, method=GET, Path=/chefs
- Create a new Target Endpoint, name=baasTarget, type=Target
  - URL=https://api.usergrid.com/bellevue2015/sandbox
- Change default Target Endpoint
  - URL=https://apigee-edu-test.apigee.net/v1/apieatery
- Setup Route Rules to send your request
  - In Proxy EndPoint, Define route rule as follows:

```
<RouteRule name="baasTarget">
    <Condition>(proxy.pathsuffix MatchesPath &quot;/ratings&quot;) and (request.verb = &quot;GET&quot;)</Condition>
    <TargetEndpoint>baasTarget</TargetEndpoint>
</RouteRule>
<RouteRule name="default”><TargetEndpoint>default</TargetEndpoint></RouteRule>
```

Day 1 – Lab 3 – Dynamic Routing Cont.

- Using Postman, test the following urls:
  - http://orgname-envname.apigee.net/v0/apieatery/ratings
  - http://orgname-envname.apigee.net/v0/apieatery/chefs

Day 1 – Lab 4 – Advance Dynamic Routing

- Add a javascript policy
  - Display name = Dynamic Routing, Script Name = Dynamic-Routing
  - Script File = Create new script, Script Name = Dynamic-Routing.js
  - Attach Policy = check, Flow = Flow baasTarget, Target PreFlow
  - Segment = request
  - Type following javascript content:

```
Try {
        var new_target = "http://httpbin.org/get";
        context.setVariable("target.url", new_target);
} catch(e) {
        throw e;
}
```

Thank you

January 2014