apigee

Aug, 2015

Foundational Training

Advance Apigee Edge

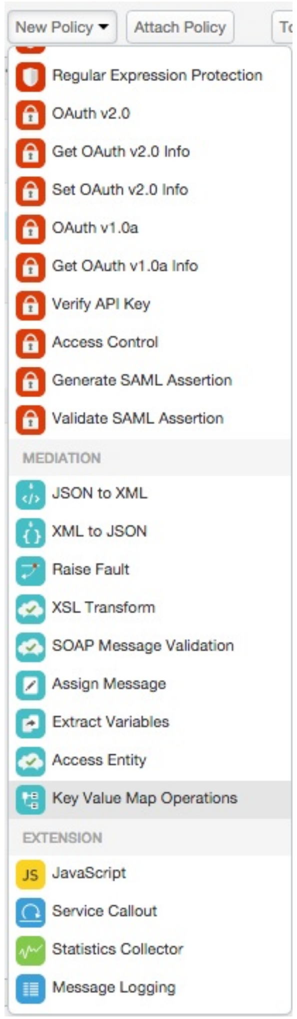
Validating Incoming Request Data

Data Validation Approaches

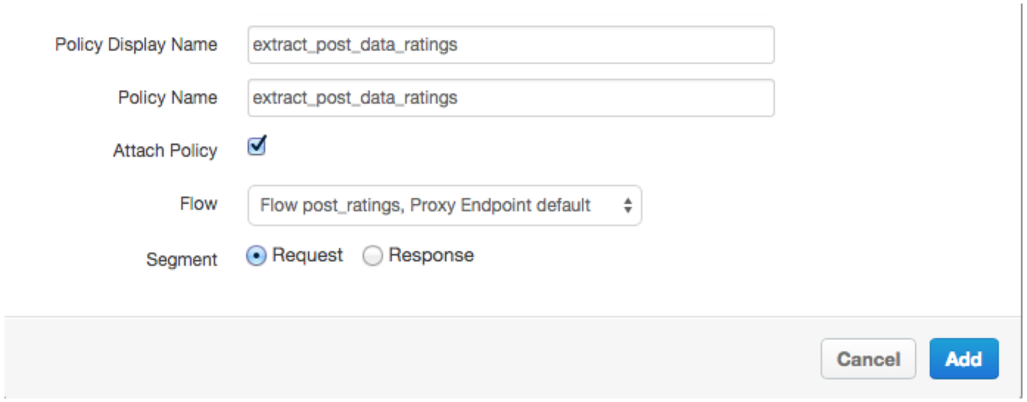
- Simple Validation
  - Extract Variables in combination with a RaiseFault Policy
    - Great for JSON, Form Parameters and simple xml structured requests
- Format Specific Validation Approaches
  - JSON
    - Javascript policies
  - XML
    - XSLT for dynamic references
  - SOAP
    - SOAP Message Validation policy
  - ALL
    - Java Callout

Simple Post Validation with Extract Variables

Add Extract Variables

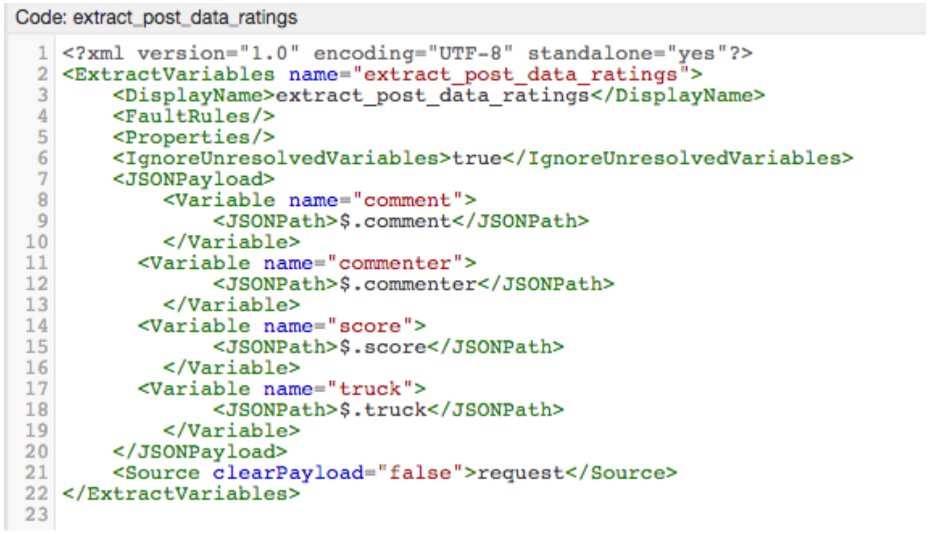


Name and Select the ProxyEndpoint (Ratings (Post)) from the flows, then press Add



Policy Display Name	extract_post_data_ratings
Policy Name	extract_post_data_ratings
Attach Policy	✓
Flow	Flow post_ratings, Proxy Endpoint default
Segment	[✓ Request] Response

Modify the content to retrieve the Post Data



Code: extract_post_data_ratings

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<ExtractVariables name="extract_post_data_ratings">
    <DisplayName>extract_post_data_ratings</DisplayName>
    <FaultRules/>
    <Properties/>
    <IgnoreUnresolvedVariables>true</IgnoreUnresolvedVariables>
    <JSONPayload>
        <Variable name="comment">
            <JSONPath>$.comment</JSONPath>
        </Variable>
      <Variable name="commenter">
            <JSONPath>$.commenter</JSONPath>
        </Variable>
      <Variable name="score">
            <JSONPath>$.score</JSONPath>
        </Variable>
      <Variable name="truck">
            <JSONPath>$.truck</JSONPath>
        </Variable>
    </JSONPayload>
    <Source clearPayload="false">request</Source>
</ExtractVariables>
```

Day 3 – Lab 14 ExtractVariable and AssignMessage Policy

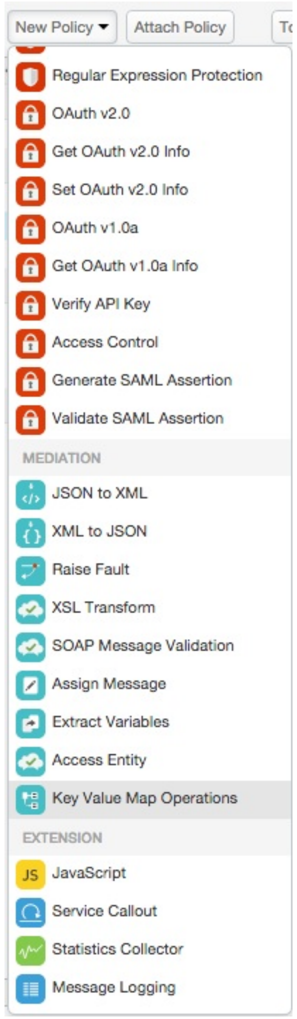
- Check that /rating is functional by sending a request
- Take a look at the response
  - Create extractMessage policy to get the following parameters from the json **RESPONSE**:
    - Application $.application
    - Comment $.entities[0].comment
    - Commenter $.entities[0].commenter
  - Send a request and run trace to make sure these flow variables are set
- Create a AssignMessage policy to build a new **RESPONSE** with above flow variables

```
<Set>
    <Payload contentType="text/plain">\{
      "application":{application},
      "comment":{comment},
      "commenter":{commenter}
      }
    </Payload>
  </Set>
```

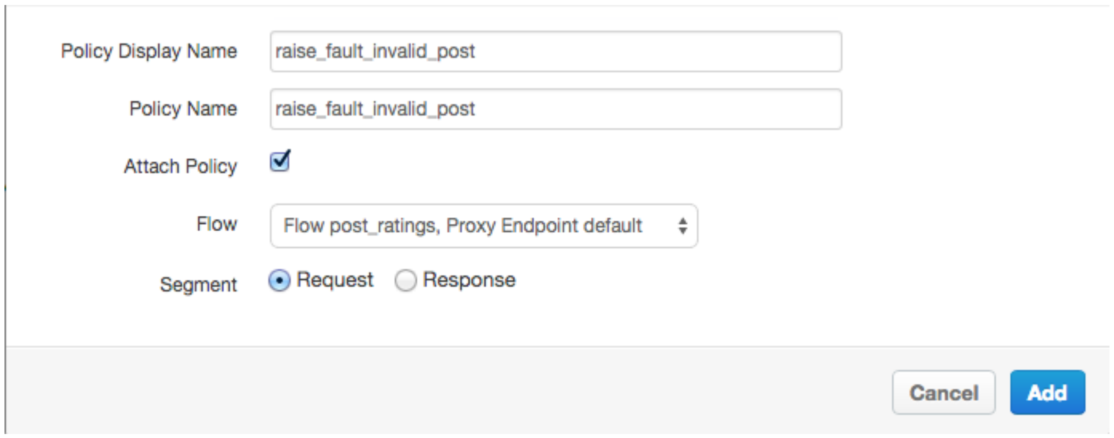
- Test your code

Raising a Fault (error)

Add a Raise Fault Policy

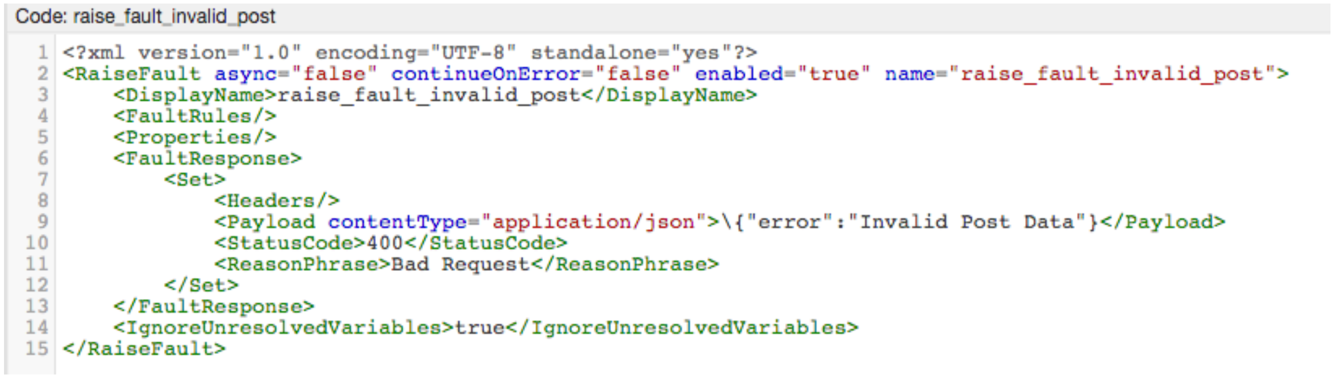


Name and Select the ProxyEndpoint (Ratings (Post)) from the flows, then press Add



Policy Display Name	raise_fault_invalid_post
Policy Name	raise_fault_invalid_post
Attach Policy	✓
Flow	Flow post_ratings, Proxy Endpoint default
Segment	[✓ Request] Response

Modify the content to return an appropriate fault message



Code: raise_fault_invalid_post

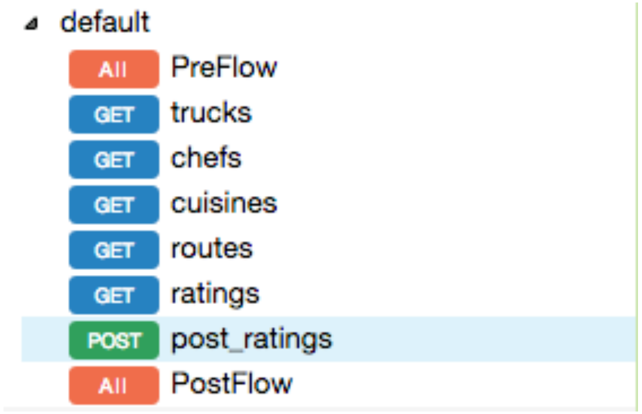
```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<RaiseFault async="false" continueOnError="false" enabled="true" name="raise_fault_invalid_post">
    <DisplayName>raise_fault_invalid_post</DisplayName>
    <FaultRules/>
    <Properties/>
    <FaultResponse>
        <Set>
            <Headers/>
            <Payload contentType="application/json">\{"error":"Invalid Post Data"}</Payload>
            <StatusCode>400</StatusCode>
            <ReasonPhrase>Bad Request</ReasonPhrase>
        </Set>
    </FaultResponse>
    <IgnoreUnresolvedVariables>true</IgnoreUnresolvedVariables>
</RaiseFault>
```

Raise Fault Policy Details

- Fault Response Tag Format
  - If empty it will redirect request / response flow to the fault rules for handling
  - Set – Allows you to build an inline response to fault
    - StatusCode – HTTP Status Code
    - ReasonPhrase – HTTP Reason Phrase
    - Payload – Message Contents for your fault (follows same spec as assign message)
  - Copy – Allows you to copy the request or the response into the fault response
    - Attr:source – request or response object
- Note : Raise Fault can also be used as a mechanism to stop the request flow before getting to the target in a success scenario.

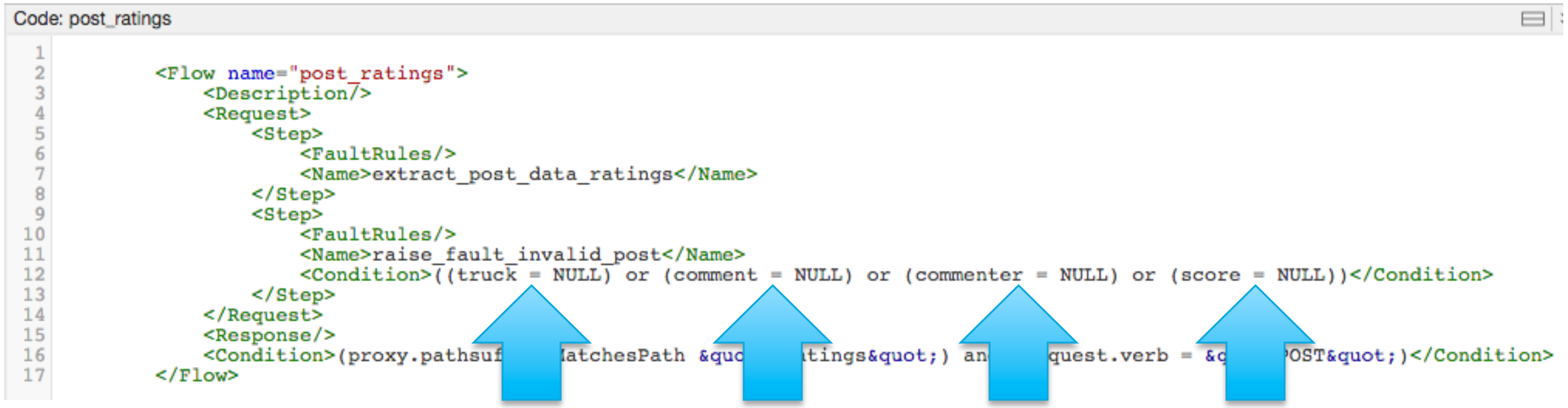
Raise Faults work with conditions

Select Post Ratings Resource from navigation pane



- default
  - All PreFlow
  - GET trucks
  - GET chefs
  - GET cuisines
  - GET routes
  - GET ratings
  - POST post_ratings
  - All PostFlow

Add a condition to the Step (policy)



Code: post_ratings

```

        <Flow name="post_ratings">
            <Description/>
            <Request>
                <Step>
                    <FaultRules/>
                    <Name>extract_post_data_ratings</Name>
                </Step>
                <Step>
                    <FaultRules/>
                    <Name>raise_fault_invalid_post</Name>
                    <Condition>((truck = NULL) or (comment = NULL) or (commenter = NULL) or (score = NULL))</Condition>
                </Step>
            </Request>
            <Response/>
            <Condition>(proxy.pathsuf     latchesPath &quo     tings&quot;) an     quest.verb = &q     OST&quot;)</Condition>
        </Flow>
```

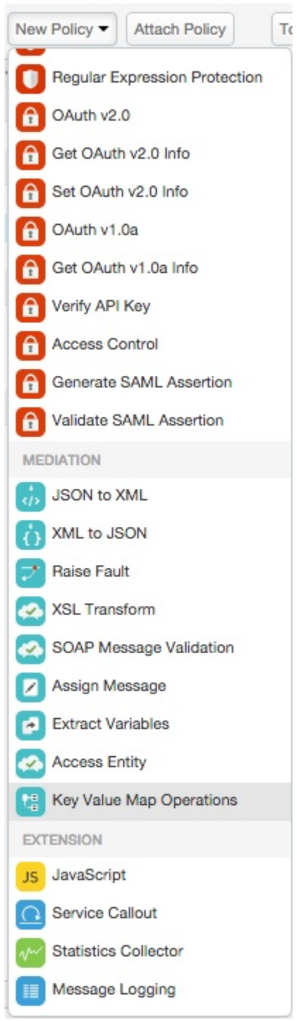
Day 3 – Lab 12 Build new resources – Pre-Mashup

- Create a new resource /trucks, method=GET
- Test your proxy using Postman
- Create a new resource /trucks/{truckname}
  - method=GET
- Using Postman, send a request to /trucks
  - Note the truck name in the response
- Using Postman, send a request to /trucks/{truckname}
  - Provide the truck name in the URL (without the braces)

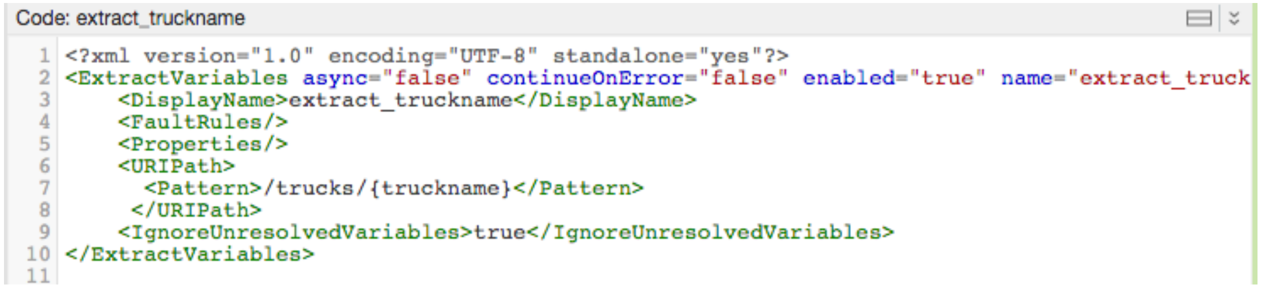
Creating a Mashup

Extracting URL Id's

Start By Creating a Extract Variables



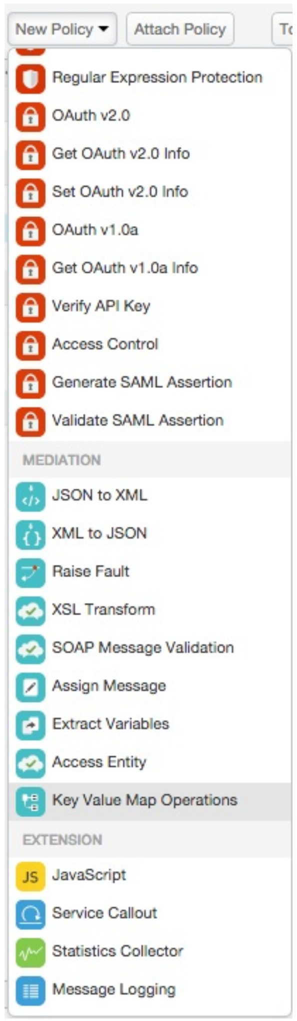
Modify the content to extract the Truck Name from the request


Code: extract_truckname

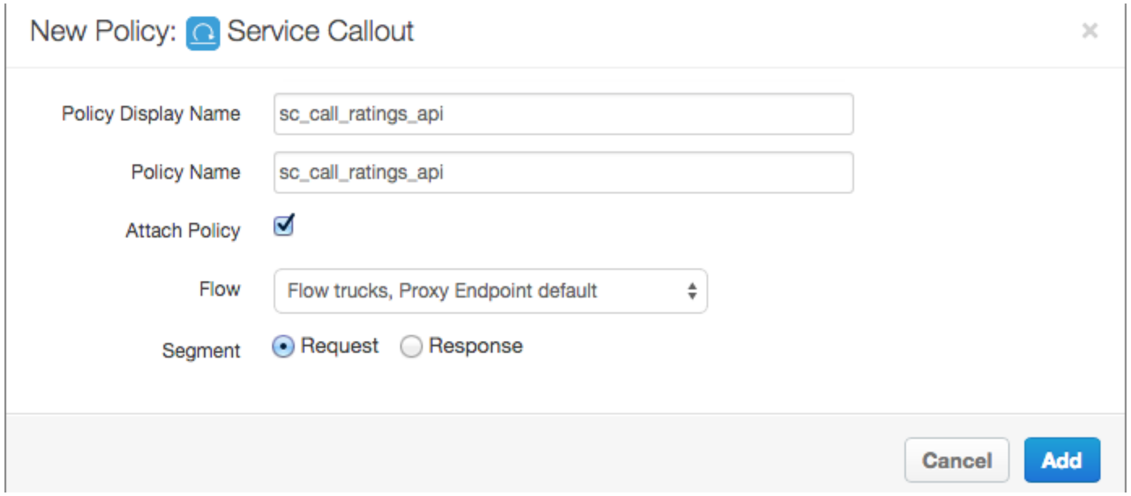
```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<ExtractVariables async="false" continueOnError="false" enabled="true" name="extract_truck
    <DisplayName>extract_truckname</DisplayName>
    <FaultRules/>
    <Properties/>
    <URIPath>
      <Pattern>/trucks/{truckname}</Pattern>
     </URIPath>
    <IgnoreUnresolvedVariables>true</IgnoreUnresolvedVariables>
</ExtractVariables>

```

Create a Service Callout

Start By Creating a Service Callout



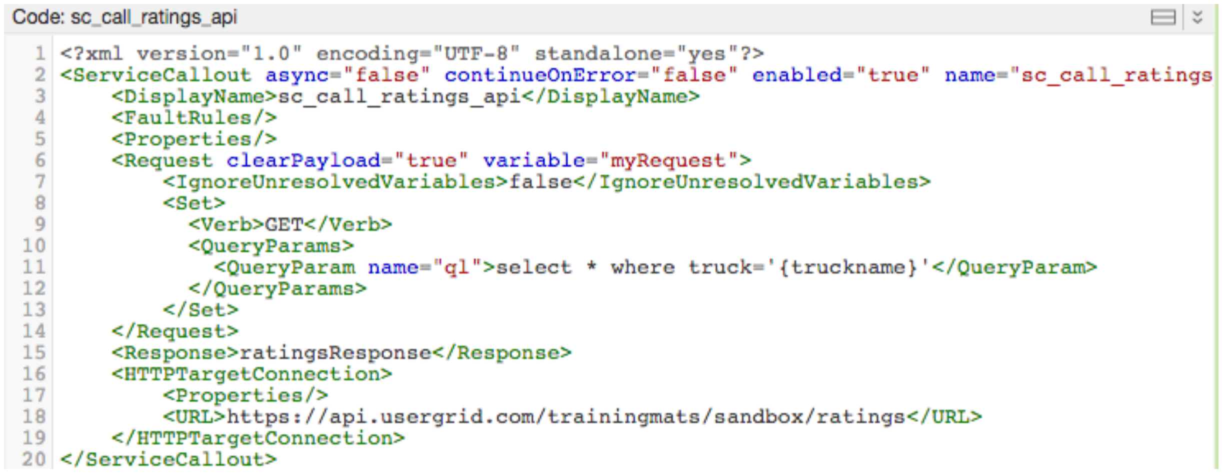
Name and Select the ProxyEndpoint (Trucks(Get)) from the flows, then press Add



New Policy: Service Callout

Policy Display Name	sc_call_ratings_api
Policy Name	sc_call_ratings_api
Attach Policy	✓
Flow	Flow trucks, Proxy Endpoint default
Segment	[✓ Request] Response

Modify the content to return an appropriate request message

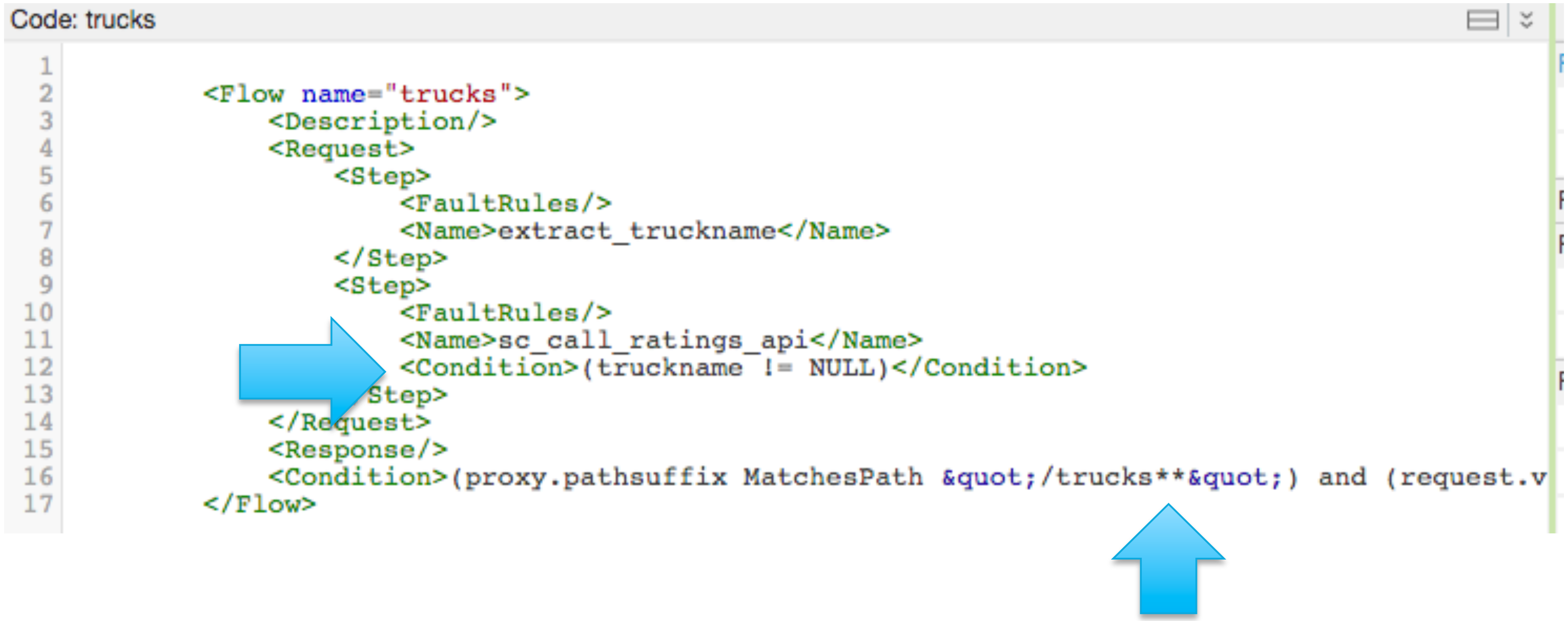


Code: sc_call_ratings_api

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<ServiceCallout async="false" continueOnError="false" enabled="true" name="sc_call_ratings
    <DisplayName>sc_call_ratings_api</DisplayName>
    <FaultRules/>
    <Properties/>
    <Request clearPayload="true" variable="myRequest">
        <IgnoreUnresolvedVariables>false</IgnoreUnresolvedVariables>
        <Set>
          <Verb>GET</Verb>
          <QueryParams>
            <QueryParam name="ql">select * where truck='{truckname}'</QueryParam>
          </QueryParams>
        </Set>
    </Request>
    <Response>ratingsResponse</Response>
    <HTTPTargetConnection>
        <Properties/>
        <URL>https://api.usergrid.com/trainingmats/sandbox/ratings</URL>
    </HTTPTargetConnection>
</ServiceCallout>
```

Add some extra conditions

Add A Condition to the service callout, so that it only runs when a truck detail request is made



```
        <Flow name="trucks">
            <Description/>
            <Request>
                <Step>
                    <FaultRules/>
                    <Name>extract_truckname</Name>
                </Step>
                <Step>
                    <FaultRules/>
                    <Name>sc_call_ratings_api</Name>
                    <Condition>(truckname != NULL)</Condition>
                </Step>
            </Request>
            <Response/>
            <Condition>(proxy.pathsuffix MatchesPath &quot;/trucks**&quot;) and (request.v
        </Flow>
```

Add ** wildcard to the end of the resource path to allow all traffic coming into trucks to pass-through to the target

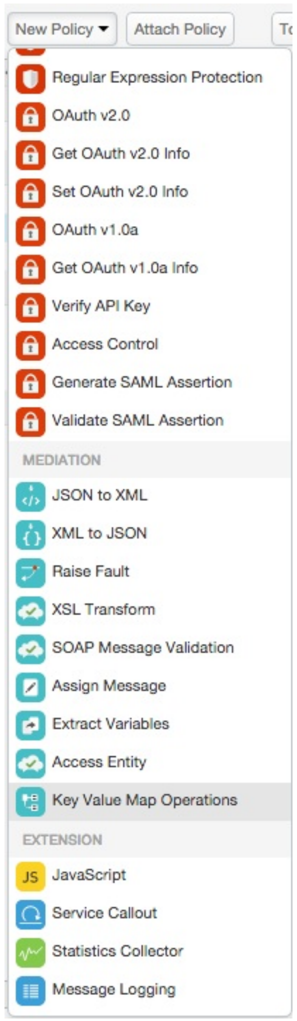
Mashing up the Responses

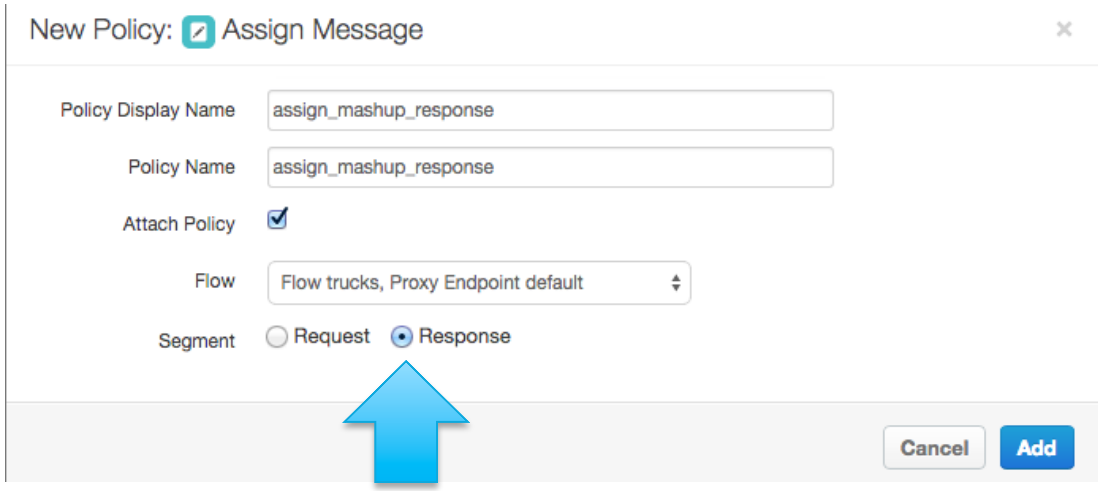
- Strategies for Mashups
  - AssignMessage (simple, most constrictive)
  - ExtractVariables, AssignMessage (simple, constrictive)
  - Javascript (flexible, complex)
  - Java (flexible, complex)
  - XSLT (flexible, complex)

  - Choose a format that works for you!!

Mashup

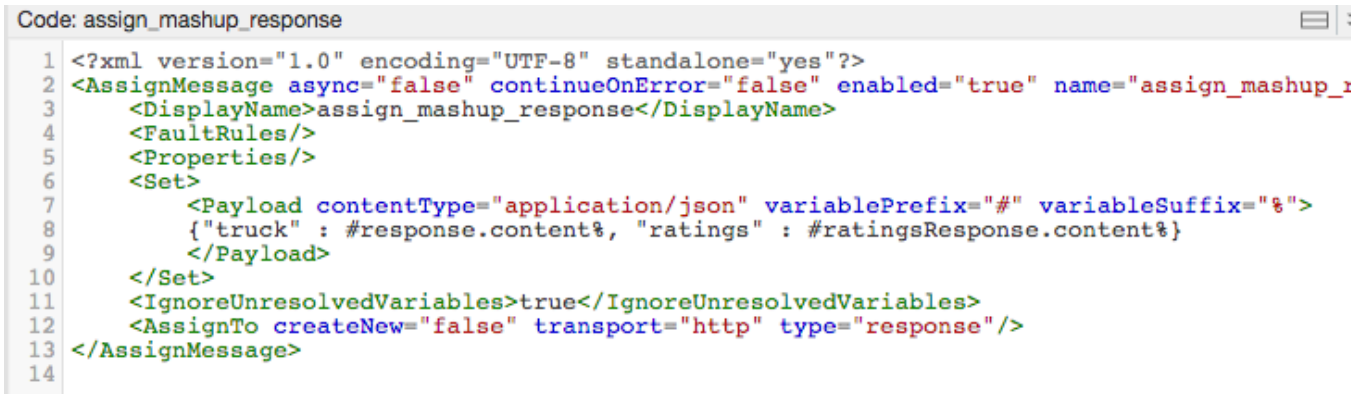
Lets Start by Creating an Assign Message Policy





New Policy: Assign Message

Policy Display Name	assign_mashup_response
Policy Name	assign_mashup_response
Attach Policy	✓
Flow	Flow trucks, Proxy Endpoint default
Segment	Request [✓ Response]

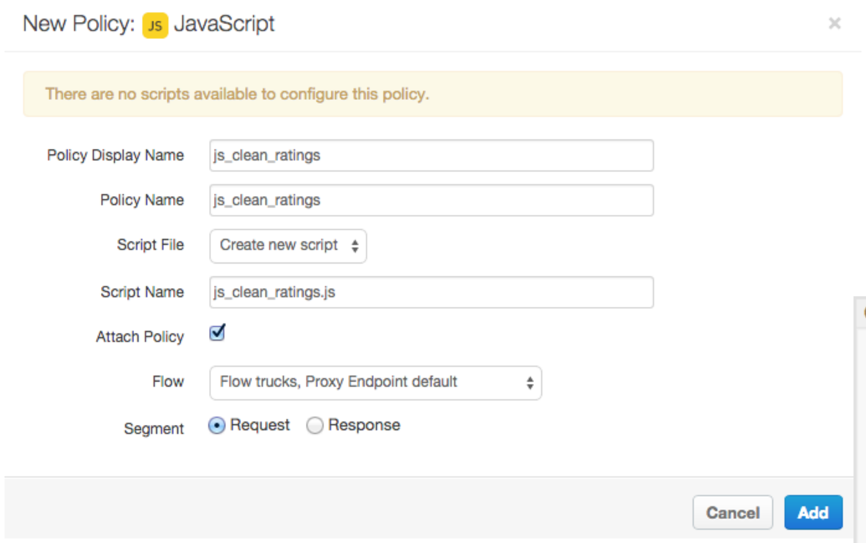


Code: assign_mashup_response

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<AssignMessage async="false" continueOnError="false" enabled="true" name="assign_mashup_r
    <DisplayName>assign_mashup_response</DisplayName>
    <FaultRules/>
    <Properties/>
    <Set>
        <Payload contentType="application/json" variablePrefix="#" variableSuffix="%">
        {"truck" : #response.content%, "ratings" : #ratingsResponse.content%}
        </Payload>
    </Set>
    <IgnoreUnresolvedVariables>true</IgnoreUnresolvedVariables>
    <AssignTo createNew="false" transport="http" type="response"/>
</AssignMessage>
```

A Mashed Up Response Sure but…..

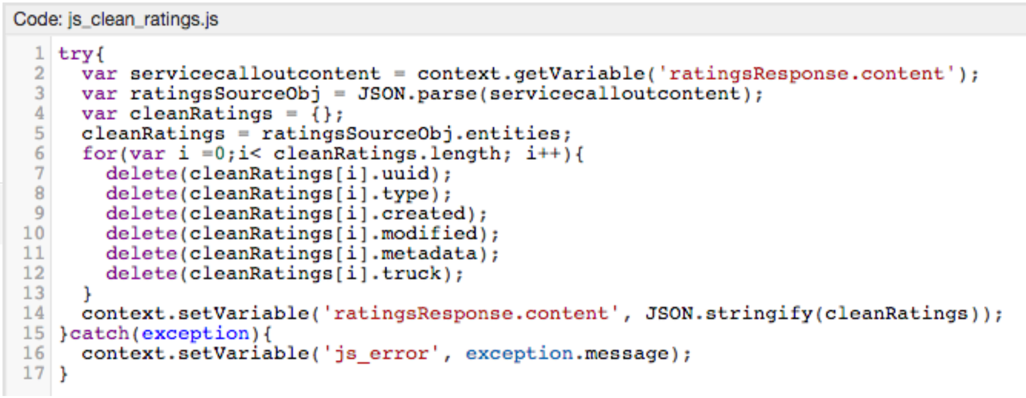
- There is a lot of extra metadata we don't care about in the ratings response
- On method of cleansing your json of unwanted fields is to use the JavaScript policy, or you could extract out the fields you want with an extract variables policy, or a combination of both.


New Policy: JavaScript

There are no scripts available to configure this policy.

Policy Display Name	js_clean_ratings
Policy Name	js_clean_ratings
Script File	Create new script
Script Name	js_clean_ratings.js
Attach Policy	✓
Flow	Flow trucks, Proxy Endpoint default
Segment	[✓ Request] Response

Note: you will have copy the condition from the service callout and apply it to the js policy too


Code: js_clean_ratings.js

```
try{
  var servicecalloutcontent = context.getVariable('ratingsResponse.content');
  var ratingsSourceObj = JSON.parse(servicecalloutcontent);
  var cleanRatings = {};
  cleanRatings = ratingsSourceObj.entities;
  for(var i =0;i< cleanRatings.length; i++){
    delete(cleanRatings[i].uuid);
    delete(cleanRatings[i].type);
    delete(cleanRatings[i].created);
    delete(cleanRatings[i].modified);
    delete(cleanRatings[i].metadata);
    delete(cleanRatings[i].truck);
  }
  context.setVariable('ratingsResponse.content', JSON.stringify(cleanRatings));
}catch(exception){
  context.setVariable('js_error', exception.message);
}
```

Day 3 – Lab 12 ServiceCallout and Mashup Cont.

- Create a ExtractVariable policy
  - Attach to truckDetail Proxy Endpoint
  - segment=request

```
<URIPath><Pattern>/trucks/{truckname}</Pattern></URIPath>
<IgnoreUnresolvedVariables>true</IgnoreUnresolvedVariables>
<Source clearPayload="false">request</Source>
```

  - Test in postman and trace to ensure “truckname” is captured in flow variables
- Create a ServiceCallOutPolicy
  - attach to truckDetail Proxy Endpoint
  - segment=request

Day 3 – Lab 12 ServiceCallout and Mashup Cont.

```
<Request clearPayload="true" variable="myRequest">
    <Set>
        <Verb>GET</Verb>
        <QueryParams>
          <QueryParam name="ql">select * where truck = '{truckname}'</QueryParam>
        </QueryParams>
    </Set>
    <IgnoreUnresolvedVariables>false</IgnoreUnresolvedVariables>
 </Request>
 <Response>ratingsResponse</Response>
 <HTTPTargetConnection>
    <Properties/>
    <URL>https://api.usergrid.com/bellevue2015/sandbox/ratings</URL>
 </HTTPTargetConnection>
```

Day 3 – Lab 12 ServiceCallout and Mashup Cont.

- Create a AssignMessage Policy
  - Attach to Flow truckDetail, Proxy Endpoint default
  - Segment = Response

```
<Set>
      <Payload contentType="application/json" variablePrefix="%" variableSuffix="#">{"truck":
%response.content#, "ratings":%ratingsResponse.content#}</Payload>
 </Set>
 <IgnoreUnresolvedVariables>true</IgnoreUnresolvedVariables>
 <AssignTo createNew="false" transport="http" type="response"/>
```

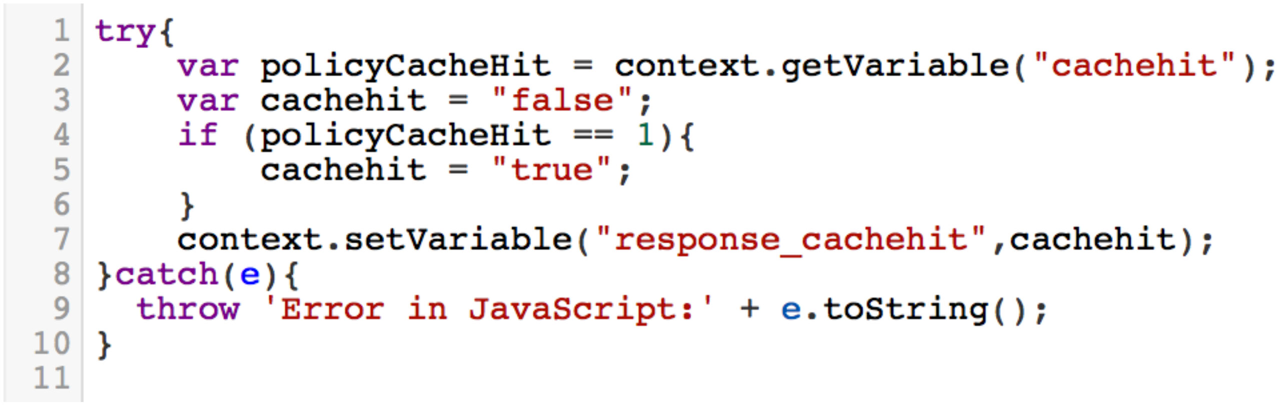
Extend with Programming

Extreme Flexibility with JavaScript, Java and Python

When you need more flexibility than supported by the out-of-the box policies, Edge has the ability for a developer to use server-side JavaScript, Java, or Python.

- External libraries can be included at the organization level or via proxy. This is defined with <IncludeURL> defined in the policy.
- Execution time limits available in policy to avoid infinite loop or slow performing code.

JavaScript Overview



```
try{
    var policyCacheHit = context.getVariable("cachehit");
    var cachehit = "false";
    if (policyCacheHit == 1){
        cachehit = "true";
    }
    context.setVariable("response_cachehit",cachehit);
}catch(e){
  throw 'Error in JavaScript:' + e.toString();
}
```

Allows you to run server-side JavaScript to extend the capability of proxy processing. Important when needing to use loops/switches/complex logic.

- Preferred choice of callout amongst Apigee developers
- Can be used to leverage asynchronous httpclient requests
- Relies on Rhino
- Apigee uses E4X, extending capability for XML support

JavaScript Object Model

Context

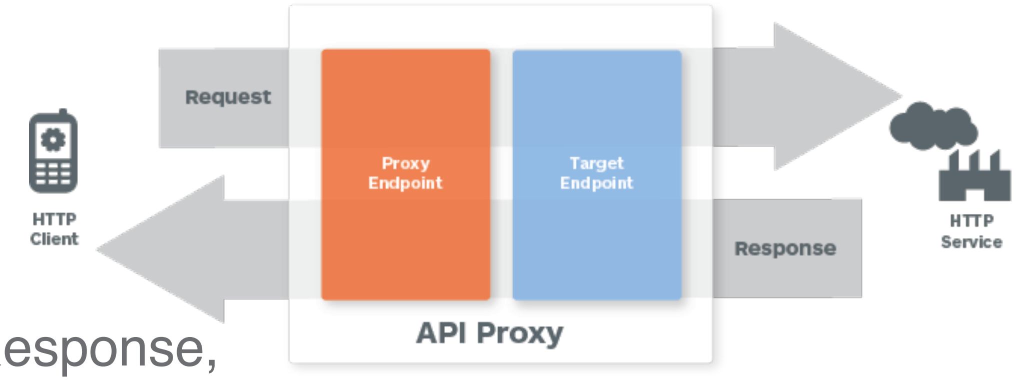
A context object (context) is created for each request/response. Within the context, you can access variables and the HTTP message.

Methods

- getVariable(), setVariable() – *context.getVariable("myVar");*

Messages Objects

- request, response
- proxyRequest, proxyResponse, targetRequest, targetResponse



JavaScript Object Model (cont'd)

Message Object Properties

The HTTP message objects contain properties for each part of the HTTP Message:

- headers
- queryParameters
- method
- body/content

JavaScript httpClient

The JavaScript http client can be used to make asynchronous http requests.

Two methods exposed:

- get()
- send()

Each method returns an exchange object that exposes additional methods:

- isError()
- isSuccess()
- isComplete()
- waitForComplete()
- getResponse()
- getError()

ServiceCallout and Mashup

DEMO

Cleaned up Truck Detail Response

Using JavaScript

Java Callout

Same principle as JavaScript, but using the java programming language. Java is typically used when needing extreme performance with complex logic.

- Not available in free Apigee organizations.

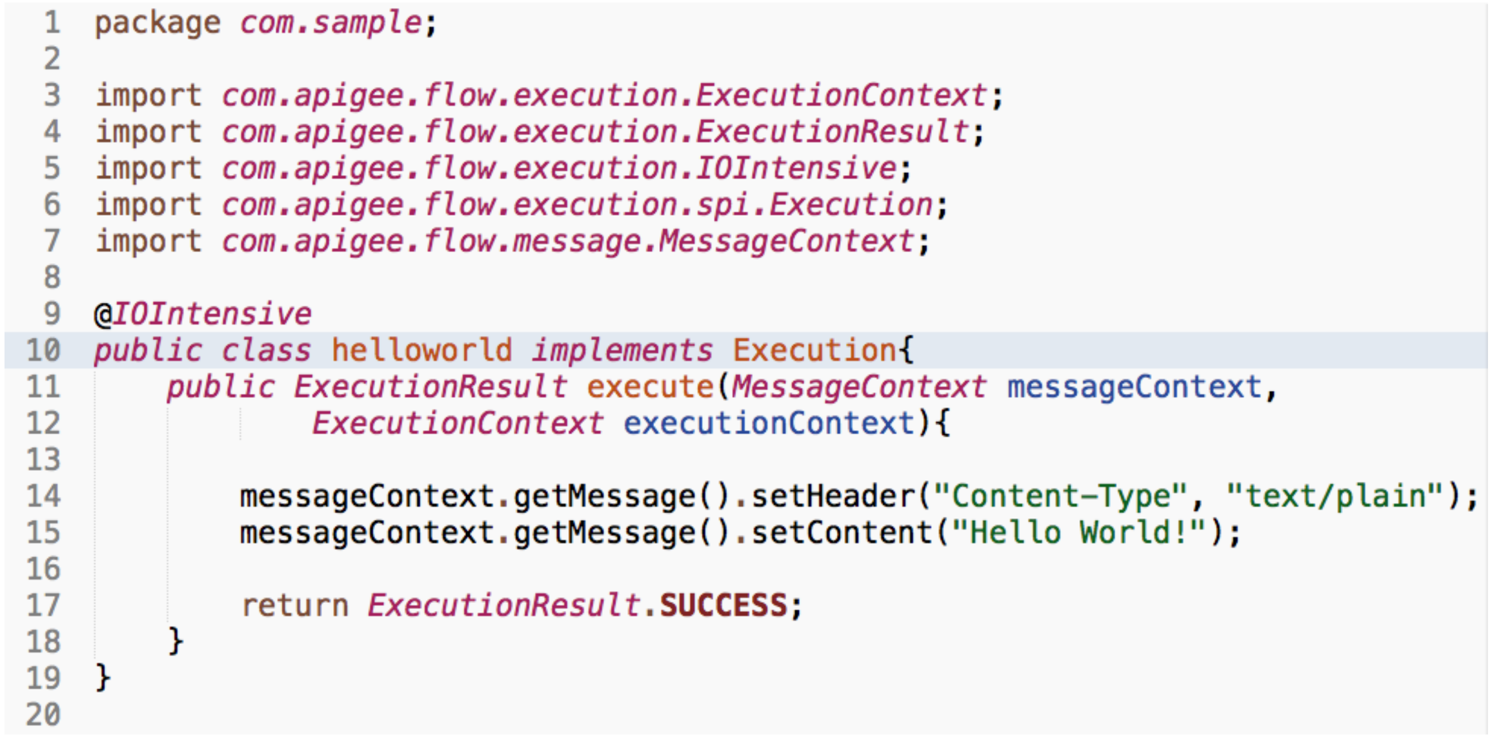
Relies on two libraries:

- expressions-1.0.0.jar
- message-flow-1.0.0.jar

https://github.com/apigee/api-platform-samples/tree/master/doc-samples/java-cookbook

Network I/O, file system read/writes, current user info, process list, and CPU/memory utilization are not permitted by the security model.

Java Callout (cont'd)


```java
package com.sample;

import com.apigee.flow.execution.ExecutionContext;
import com.apigee.flow.execution.ExecutionResult;
import com.apigee.flow.execution.IOIntensive;
import com.apigee.flow.execution.spi.Execution;
import com.apigee.flow.message.MessageContext;

@IOIntensive
public class helloworld implements Execution{
    public ExecutionResult execute(MessageContext messageContext,
            ExecutionContext executionContext){

        messageContext.getMessage().setHeader("Content-Type", "text/plain");
        messageContext.getMessage().setContent("Hello World!");

        return ExecutionResult.SUCCESS;
    }
}

```

- Be sure to import the necessary apigee libraries. When compiling the jar for upload into Apigee, however, do not include these as they already existing within the platform.

Python

Python can also be used in the same fashion as JavaScript, but with python programming language. Python is the least preferred option amongst Apigee developers.

- Not available in free Apigee organizations

Context

A context object is available for the request (request) or response (response). A separate context (flow) is available for variable access.

Variables (flow)

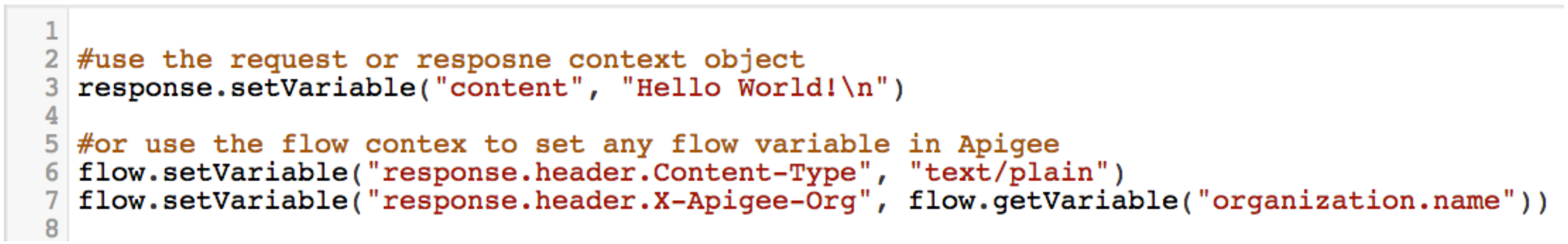
- flow.getVariable()
- flow.setVariable()

Python (cont'd)

Messages

request.getVariable("content")

response.setVariable("content")



```
#use the request or resposne context object
response.setVariable("content", "Hello World!\n")

#or use the flow contex to set any flow variable in Apigee
flow.setVariable("response.header.Content-Type", "text/plain")
flow.setVariable("response.header.X-Apigee-Org", flow.getVariable("organization.name"))
```

Choosing Which Extension Policy to Use…

Use JavaScript:

- mashing up responses/manipulating json and non-complex XML
- looping /switching through datasets
- if it’s more intuitive than Apigee policies (for example, when setting target.url for many different URI routing combinations)
- most preferred option for Apigee developers

Use Java:

- if performance is the highest priority
- when the solution requires functionality that is best served in Java (e.g. email notification service)

Use Python:

- only if absolutely required as Python scripts can introduce performance bottlenecks for simple executions since interpreted at runtime
- least preferred option for Apigee developers

DEMO Javascript mashup

DEMO

Javascript Mashup

With Google Maps

Thank you

apigee

January 2014