**Getting started with Custom Policies development and Test locally using Anypoint Studio .**

**Objective :- Educate Prepaid team to test new / existing custom policies in local Anypoint studio / on-premise Mule runtimes.**

**I have written this article to help developers learn how they could do this step by step.**

**Assumption :- Person is aware of what is Anypoint Studio , Custom policy , Mule Flows etc.**

**I have created two artifacts here .**

1. **Mule Test API :- This is a test API to apply the policy against.**
2. **Creating the custom policy .**

**1.**

|  |
| --- |
|  |

**The Mule Test API is a simple API that accepts get request to a resource called /test.**

**It returns the following JSON payload every time a GET request is being made to the /test resource.**

**{**

**“message” : “test called”**

**}**

|  |
| --- |
| <api-platform-gw:api apiName=*"testapi"* version=*"1.0"* flowRef=*"api-main"* create=*"true"* apikitRef=*"apiConfig"* doc:name=*"****API Autodiscovery****"*/> 🡪 1.  <flow name=*"api-main"*>  <http:listener config-ref=*"httpListenerConfig"* path=*"/api/\*"* doc:name=*"HTTP"* />  <apikit:router config-ref=*"apiConfig"* doc:name=*"APIkit Router"* />  <exception-strategy ref=*"apiKitGlobalExceptionMapping"* doc:name=*"Reference Exception Strategy"* />  </flow>  <flow name=*"api-console"*>  <http:listener config-ref=*"httpListenerConfig"* path=*"/console/\*"* doc:name=*"HTTP"* />  <apikit:console config-ref=*"apiConfig"* doc:name=*"APIkit Console"* />  </flow>  <flow name=*"get:/test:apiConfig"*>  <expression-filter expression=*"#[message.inboundProperties['http.query.params']['vaibhav'] contains 'vaibhav']"* doc:name=*"Expression"*/> <!— Here Line 22 -->  <set-payload value=*"{*  *&quot;message&quot;: &quot;test called&quot;*  *}"* doc:name=*"Set Payload"* mimeType=*"application/json"*/>  <logger message=*"test called "* level=*"INFO"* doc:name=*"Logger"*/>  </flow> |

**.It returns the following JSON payload every time a GET request is being made to the /test resource. On succsess**

<http://127.0.0.1:8081/api/test/vaibhav=vaibhav>

**{ “message” : “test called” }**

**Above flow will validate queryparam Vaibhav=Vaibhav.**

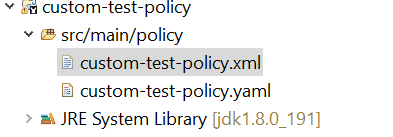
**Section B**

**Creating the custom policy using Anypoint studio . Which will interpret incoming request to /test API.**

**Step 1 ) Go to Menu 🡪File –>New -> Api custom policy project. 🡪 Provide Project Name and Select Runtime “Mule Server 3.8.4.EE”**

****

**Step 2)**

****

**The custom policy is a simple one, it intercepts request going to the “Test API” via the API gateway (Same as Sprint Architecture) and decides if it will honor the request and pass it through. Figure below shows the policy XML that has been created for this demo. There is logically 3 sections in the file, from top to bottom as depicted by the picture is**

1. **pointcut (controls the scope of a policy application You can use different types of conditions to determine if the policy must be applied.)**

**B) The entry point of the policy, where it intercepts request going though the API GATEWAY .**

**C) the processing logic - This is the section that allows a developer to declare complex logic**

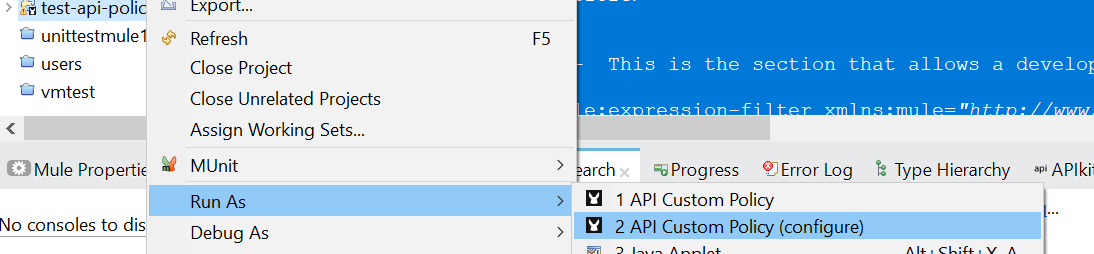
|  |
| --- |
| <!-- POINTCUT DEFINES WHERE A POLICY IMPLEMENTS Section A -->  <pointcut>  <resource uriTemplateRegex=*"/test.\*"* />  <resource methodRegex=*"GET"* />  </pointcut>  <pointcut>  <app regex=*".\*testapi.\*"* />  </pointcut>  <!-- THE ENTRY POINT OF THE POLICY, WHERE IT INTERCEPTS REQUEST GOING THROUGH THE API GATEWAY Section B -->  <before>  <mule:message-filter xmlns:mule=*"http://www.mulesoft.org/schema/mule/core"* onUnaccepted=*"policyViolation"*>  <mule:filter ref=*"Expression"*/> <!-- Expression Refer line number (22) in testapi.xml -->  </mule:message-filter>  </before>    <!-- This is the section that allows a developer to declare complex logic -->    <mule:expression-filter xmlns:mule=*"http://www.mulesoft.org/schema/mule/core"* expression=*"#[message.inboundProperties['http.query.params']['vaibhav'] contains 'vaibhav']"*  name=*"Expression"*/>    <!-- This section builds response messages when the policy fails. -->  <mule:processor-chain xmlns:mule=*"http://www.mulesoft.org/schema/mule/core"* name=*"policyViolation"*>  <!-- Set the HTTP status code to 403: -->  <mule:set-property propertyName=*"http.status"* value=*"403"*/>  <mule:set-property propertyName=*"Content-Type"* value=*"application/json"*/>  <!-- Set the payload to the description of the violation: -->  <mule:set-payload value=*"You … shall not pass MR !"*/>  </mule:processor-chain>  </policy> |

**The policy simply interrogates the “vaibhav” HTTP query parameter. If the query parameter contains the value “vaibhav,” then the policy will allow the intercepted request to be passed onward to the “Test API,” else it will return a message “You … shall not pass MR! ”**

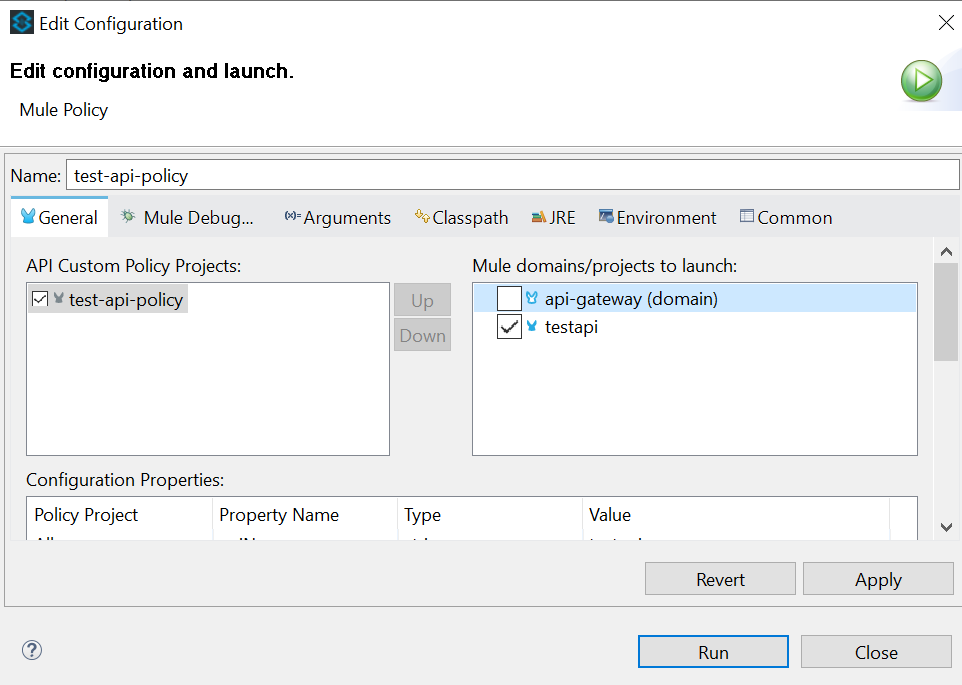
**--------------------------------------------------------------------------------------------------------------------------------------**

**Section C :-**

**Step 1 ) Right Click on “custom policy project” – Run As – API Custom Policy (configure) - Please take a look at image below.**

****

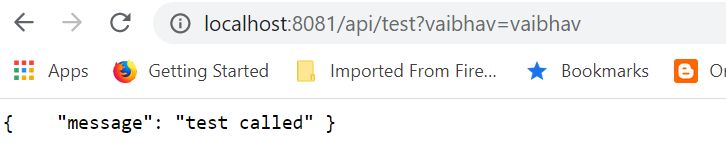
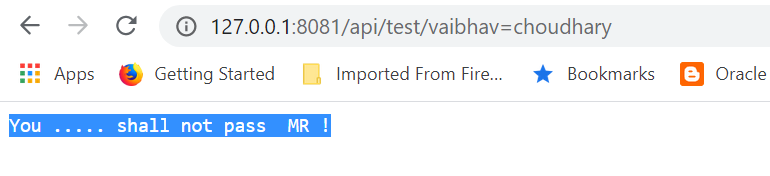
**Step 2)**

****

**Select Run Button \_ After successful deployment .**

**Execute using local browser**

<http://127.0.0.1:8081/api/test/vaibhav=choudhary>

****