whitehorses

member of the human network

WHITEBOOK

Mule Custom Policy

November 2018

Auteur:
Peter Holtland
INTEGRATIESPECIALIST

Samenvatting

MuleSoft biedt bepaalde ingebouwde policies die kunnen worden gebruikt om algemene situaties aan te pakken met betrekking tot de veiligheid of om ongewenst verkeer naar uw API te filteren, berichten te monitoren of inkomende calls te loggen.
De standaard policies die geleverd worden zijn niet altijd toereikend voor de gewenste veiligheid of voor de gewenste toepassing. Op eenvoudige manier kan een policy worden aangemaakt om de gewenste functionaliteit te krijgen. Verschillende talen worden ondersteund, maakt het aanmaken van custom policies toegankelijk. Een uitgewerkt voorbeeld is onderdeel van dit whitebook.

Inleiding

MuleSoft biedt bepaalde ingebouwde policies die kunnen worden gebruikt om algemene situaties aan te pakken met betrekking tot de veiligheid of om ongewenst verkeer naar uw API te filteren, berichten te monitoren of inkomende calls te loggen.

MuleSoft biedt de mogelijkheid om deze policies aan te passen aan de wensen en functionaliteit die nodig is. Aanpassingen in custom policies kunnen wenselijk zijn om ongewenst gedrag te onderscheppen of juist om gewenst gedrag te forceren.

Voorbeelden zijn:

- Rate Throttling
- Monitoring
- Message logging
- Veiligheid aanscherpen

Case

In dit artikel gaan we een custom policy aanmaken in MuleSoft Anypoint 3 met beperkte functionaliteit om aan te geven hoe een dergelijke custom policy opgebouwd kan worden. Verderop wordt aangetoond aan de hand van enkele voorbeelden hoe krachtig en diepgaand deze policy aanpassingen kunnen zijn.

Er wordt niet beschreven hoe een custom policy aan een API gekoppeld moet worden. Handleidingen hiervoor zijn op de Mulesoft site beschikbaar.

Configuratie template

Voor het maken van een custom policy zijn twee types configuratiebestanden vereist. Een YAML-bestand met de definitie van de nieuwe policy en xml-bestand waarin de configuratie beschreven wordt.

Mulesoft Definition-bestand - Bestand waarin de kenmerken van de custom policy worden gedefinieerd in YAML notatie. Kenmerken worden gedefinieerd met behulp van:

- name - naam van de policy
- description - beschrijving van de policy
- category - tekst gebruikt in API-platform om beleid te vermelden
- standalone - boolean om aan te geven of de policy op zichzelf staat dan wel onderdeel is van een andere policy, dit kan bijvoorbeeld betekenen dat de policy niet aleen gebruikt kan worden.
- configuration - definitie van parameters voor de policy

MuleSoft Configuration-bestand (XML) - Bestand met de daadwerkelijke logica/ implementatie van de gedefinieerde policy.

Er zijn twee hoofdtags:

<before> - Code geschreven in de <before> -tag wordt uitgevoerd bij elk request dat binnenkomt bij de API voordat toegang verleend wordt aan de service.

<after> - De <after>-tag wordt uitgevoerd na voltooiing van de API-service, d.w.z. nadat het request is afgehandeld.

Als voorbeeld gebruiken we een custom policy om response caching te implementeren.

Eerst de YAML:

Hier worden 2 properties geconfigureerd die we nodig hebben voor de caching.

TTL, bepaalt hoe lang een cache-entry geldig is, defaultwaarde staat op 60000 milliseconden, minimum is 0, betekent dat de cache eigenlijk uitstaat maar de policy is nog wel actief.

Key, bepaalt wat het cache-token is waarmee de identiteit van de cach-entry aangegeven wordt.

Wanneer de waarde die hoort bij de Key, niet gevonden wordt, dan gaat het request door de normale backend.

Type van de property Key is een expression, dit betekent dat met behulp van de Key een waarde wordt bepaald, het cache-token wordt bepaald door het Key-statement.

Beide properties zijn verplicht voor de policy en moeten ingevuld worden wanneer de policy gekoppeld wordt aan een API.

Hieronder is de volledige YAML:

```
category: Performance
configuration:
defaultValue: 60000
Time to Live in ms. Validiteit van de data in de cache is max
defaultValue*milliseconden.”
maximumValue: 2147483647
minimumValue: 0
name: “TTL”
optional: false
propertyName: ttl
sensitive: false
wanneer deze parameter op true staat wordt de inhoud hiervan eruit
gefilterd, handig bij wachtwoorden, tokens enz...
type: int
description: “cache index expressie, aan de hand van het request wordt een
cache index waarde bepaalt”
name: “cache-key”
```

```
optional: false
propertyName: key
sensitive: false
type: expression
description: “caching response gebaseerd op de cache-key”
id: custom-caching-policy
name: “Custom Caching Policy”
providedCharacteristics:
- “Message caching”
requiredCharacteristics: []
requiresConnectivity: false
standalone: true
type: custom
```

Hieronder is het basis template voor de configuratie van een custom policy:

```
<policy>
<before>
<!-- Elements automatically executed at the start -->
</before>
<after>
<!-- Elements automatically executed at the end -->
</after>

<mule:processor-chain name=”chain1”>
<!-- This flow may be called to be executed by the others -->
</mule:processor-chain>

<mule:processor-chain name=”chain2”>
<!-- This flow may be called to be executed by the others -->
</mule:processor-chain>
</policy>
```

Custom Policy

Dit gaan we aanpassen om de functionaliteit te maken die we willen.

Belangrijkste twee onderdelen van de configuratie zijn de blokken <before> en <after>. Beide blokken zijn optioneel, maar de configuratie moet minimaal één ervan bevatten.

Anders wordt er niets uitgevoerd en gaat het request normaal door in de flow.

Datgene wat in het <before> blok staat, wordt uitgevoerd zodra een request aankomt bij de API, dus voor de stappen die gedefinieerd zijn in de flow van de API.

De inhoud in het <after> blok wordt uitgevoerd na de flow maar voordat het bericht door gerouteerd wordt naar een volgend endpoint.

<mule:processor-chain name="chain1">, <mule:processor-chain name="chain2"> blok kan gebruikt worden om verdere subflows te bouwen. Deze kunnen vervolgens in de <before> <after> blokken aangeroepen worden. Deze chains zijn geen subflows maar gedragen zich gelijkaardig, het zijn als het ware procedures die in de gedefinieerde volgorde uitgevoerd worden.

Beiden hebben synchroon gedrag, beiden erven de processing strategie.
Volgens de Mulesoft documentatie zal het gebruik van het <mule:processor-chain> in versies 4.xx niet meer nodig zijn.

Op deze manier kan er een keten aan acties gemaakt worden die eventueel afhankelijk zijn van elkaar. Het zijn een soort functies/procedures die binnen de policy aangeroepen kunnen worden.

Hieronder volgt de code van de configuratie:

```
<policy id="{{policyId}}" policyName="custom-caching-policy"
xmlns="http://www.mulesoft.org/schema/mule/policy"
xmlns:mule="http://www.mulesoft.org/schema/mule/core"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:api-platform-gw="http://www.mulesoft.org/schema/mule/api-platform-gw"
xmlns:spring="http://www.springframework.org/schema/beans"
xmlns:objectstore="http://www.mulesoft.org/schema/mule/objectstore"
xmlns:ee="http://www.mulesoft.org/schema/mule/ee/core"
```

```
xsi:schemaLocation="http://www.mulesoft.org/schema/mule/policy http://
www.mulesoft.org/schema/mule/policy/current/mule-policy.xsd http://www.
mulesoft.org/schema/mule/core http://www.mulesoft.org/schema/mule/core/
current/mule.xsd
http://www.springframework.org/schema/beans http://www.springframework.org/
schema/beans/spring-beans-current.xsd
http://www.mulesoft.org/schema/mule/ee/core http://www.mulesoft.org/schema/
mule/ee/core/current/mule-ee.xsd
http://www.mulesoft.org/schema/mule/objectstore http://www.mulesoft.org/
schema/mule/objectstore/current/mule-objectstore.xsd
http://www.mulesoft.org/schema/mule/api-platform-gw http://www.mulesoft.
org/schema/mule/api-platform-gw/current/mule-api-platform-gw.xsd">

<!-- caching responses using an key expression. -->

<spring:beans>
<spring:bean class="java.util.concurrent.locks.ReentrantLock" id="lock"/>
<spring:bean id="sn" name="Bean" class="org.mule.util.store.
ManagedObjectStore">
<spring:property name="storeName" value="myStore"/>
<spring:property name="entryTTL" value="{{ ttl }}"/>
<spring:property name="expirationValue" value="{{ ttl }}"/>
<spring:property name="maxEntries" value="-1"/>
</spring:bean>
</spring:beans>
<mule:processor-chain xmlns:mule="http://www.mulesoft.org/schema/mule/core"
name="cachedResponse">
<mule:logger message="Policy {{ policyId }} cached response for key:
#[flowVars['key']]" level="DEBUG" />
<mule:set-property propertyName="http.status" value="200"/>
<mule:set-payload value="#[app.registry['myStore'].retrieve(key)]"/>
</mule:processor-chain>
<before>
<mule:set-variable variableName="key" value="{{ key }}" />
<!-- caching payload -->
<mule:choice xmlns:mule="http://www.mulesoft.org/schema/mule/core">
<mule:when expression="#[((flowVars['key'] != null) &amp;&amp;
(keyinstanceof  java.io.InputStream))]">
<mule:set-variable variableName="key" value="#[org.apache.commons.
io.IOUtils.toString(key, 'UTF-8')]"/>
<mule:set-payload value="#[key]"/>
</mule:when>
<mule:otherwise>
<mule:logger message="Key [#[message.getId()]] is not type java.
```

```
io.InputStream" level="DEBUG"/>
</mule:otherwise>
</mule:choice>
<mule:expression-component><![CDATA[if (flowVars['key'] == null)
flowVars['key'] = "";]]></mule:expression-component>

<mule:message-filter xmlns:mule="http://www.mulesoft.org/schema/mule/core"
onUnaccepted="cachedResponse">
<mule:expression-filter xmlns:mule="http://www.mulesoft.org/schema/mule/
core" expression="#[!app.registry['myStore'].contains(key)]" name="Cache-
Token"/>
</mule:message-filter>

<mule:custom-transformer name="Object-to-inputstream" class="org.mule.
transformer.simple.ObjectToInputStream" />
</before>

<after>

<mule:choice>
<mule:when expression="#[payload instanceof org.mule.api.transport.
OutputHandler]">
<mule:custom-transformer class="org.mule.transformer.simple.
ObjectToInputStream"/>
</mule:when>
<mule:otherwise>
<mule:logger message="Response Payload [#[message.getId()]] invalid
request" level="DEBUG"/>
</mule:otherwise>
</mule:choice>
<mule:choice>
<!-- Read payload if it is of InputStream datatype -->
<mule:when expression="#[payload instanceof java.io.InputStream]">
<mule:set-variable variableName="myPayload" value="#[org.apache.commons.
io.IOUtils.toString(myPayload, 'UTF-8')]" />
<mule:expression-component><![CDATA[app.registry['lock'].lock();
if (!app.registry['myStore'].contains(key))
app.registry['myStore'].store(key, myPayload);
app.registry['lock'].unlock();]]>
</mule:expression-component>
<mule:set-payload value="#[myPayload]" />
<mule:remove-variable variableName="myPayload" />
</mule:when>
<mule:when expression="#[payload instanceof java.io.Serializable]">
```

```
<mule:expression-component><![CDATA[app.registry['lock'].lock();
if (!app.registry['myStore'].contains(key))
app.registry['sn'].store(key, myPayload);
app.registry['lock'].unlock();]]>
</mule:expression-component>
</mule:when>
<mule:otherwise>
<mule:logger level="INFO" message="Invalid payload: #[message.getId()] of
#[message.getPayload().getClass()]"/>
</mule:otherwise>
</mule:choice>
<mule:remove-variable variableName="key" />
</after>
</policy>
```

Toepassing

Voor de code blokken kunnen verschillende sources gebruikt worden. Bijvoorbeeld onder andere Java, Groovy en Dataweaver.

De mogelijkheid om policies aan te passen geeft veel opties om bijvoorbeeld de veiligheid te vergroten, standaard werkwijzen te forceren op alle gepubliceerde API's. Er zijn veel toepassingen mogelijk.

Hieronder zijn nog wat voorbeelden van custom policies die de moeite waard zijn om te bekijken.

Oauth 2.0 aanpassing

Het probleem met Oauth 2.0 is dat zolang een door de provider toegekend token geldig is, deze in opeenvolgende calls richting eenzelfde API niet meer gevalideerd wordt in de backend. Token validaties worden gecached(!!!), hier staat de code voor een policy die dit probleem oplost.

Rate Limiting

Handige policy om 'fair use' te forceren. Deze policy houdt per gebruiker bij hoe vaak een bepaalde API wordt aangeroepen. De uitwerking van de policy vindt u hier

Message logging

Simpele policy die standaard de request berichten wegschrijft naar een logbestand. Handig om te gebruiken bij API-calls, hoeft de logging niet telkens in de flow gebouwd te worden. Beschrijving en source kan hier gevonden worden.

Conclusie

Policies worden onder andere gebruikt om user-authenticatie, toegang, API-consumptie en SLA's filters toe te passen. De standaard policies die geleverd worden zijn niet altijd toereikend voor de gewenste veiligheid of voor de gewenste toepassing. Op eenvoudige manier kan een policy worden aangemaakt om de gewenste functionaliteit te krijgen. Verschillende talen worden ondersteund, maakt het aanmaken van custom policies toegankelijk.