**Weather API REST adapter**

**Nov 11, 2020**

**Version 1**

| REVISION HISTORY | | | |
| --- | --- | --- | --- |
| DATE | VERSION | DESCRIPTION | AUTHOR |
| Nov 11, 2020 | 1 | Technical Specification document for weather mule application | Sachin Arora |

Table of Contents

[Weather API REST adapter 1](#_Toc56953947)

[Nov 11, 2020 1](#_Toc56953948)

[Version 1 1](#_Toc56953949)

[INTRODUCTION 3](#_Toc56953950)

[PURPOSE 3](#_Toc56953951)

[INTENDED AUDIENCE AND PERTINENT SECTIONS 3](#_Toc56953952)

[PROJECT SCOPE 3](#_Toc56953953)

[DESCRIPTION 4](#_Toc56953954)

[PRODUCT PERSPECTIVE 4](#_Toc56953955)

[FEATURES 4](#_Toc56953956)

[USER OVERVIEW 4](#_Toc56953957)

[OPERATING ENVIRONMENT 4](#_Toc56953958)

[ASSUMPTIONS / DEPENDENCIES 4](#_Toc56953959)

[ARCHITECTURE AND DEPLOYMENT 5](#_Toc56953960)

[RAML :- 6](#_Toc56953961)

[CHALLENGES 8](#_Toc56953962)

[BACKLOG and FUTURE SCOPE 8](#_Toc56953963)

# INTRODUCTION

## PURPOSE

The purpose of this document is to explain the technicalities of the REST adapter written for a legacy web-service.

## INTENDED AUDIENCE AND PERTINENT SECTIONS

This document is intended for a technical audience that will use the web-service adapter as a means to wrap for modernization of their Weather SOAP web-service.

## PROJECT SCOPE

Weather rest adapter is working as an interface for a legacy SOAP based web service. This application exposes the services as a REST API by giving meaningful and simpler integration. Any kind of future requirement can easily be incorporated by updating the mule flow of the application.

# DESCRIPTION

## PRODUCT PERSPECTIVE

Originally as a proof of concept for clients, this work was born out of our knowledge in integration tools; integration is still as relevant as fifteen years ago, when the SOA initiatives were being seen as rather disappointing. Integration is wide and convoluted area of software intensive systems that needs a keen eye and good amount of experience and expertise. We proud ourselves in having both and it is out opinion that the approach to enterprise-wide integration from a cultural and process perspectives is more important than tools, but tools, ultimately are what end up materializing the view of business processes.

MuleSoft has achieved a good name and its ESB product is becoming a very important player in the market. Although proprietary, it offers concrete and effective ways to integrate in an agile way. This service adapter is proof of that.

## FEATURES

The client has a Weather SOAP-based web-service, and through Mule ESB and Anytime Studio IDE, a RESTful adapter was done in order to simplify its consumption, from the perspective of simpler interfaces, much less verbose output and input and maintainability as well as modifiability.

## USER OVERVIEW

The user is typically another actor system, a client that consumes HTTP endpoints with URI parameters and using the GET HTTP method. This would be accomplished through POSTMAN or other REST client.

## OPERATING ENVIRONMENT

As long as the final user, which is a technical user, has Node and Mule runtime 4.2.0 installed, the modified SOAP code (more on this below), this service adapter will work normally. The original resource given by the client for their SOAP-based Weather web-service was offline almost a year ago, so I used the Node version to run it locally. Some modifications were in order to mock the responses of the original Weather service.

## ASSUMPTIONS / DEPENDENCIES

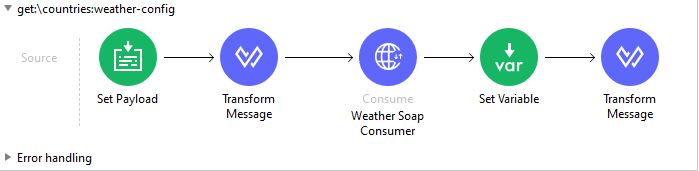
The original SOAP-based Weather service must be running locally on localhost and port 8080, as the source code, given by the client, indicates.

## ARCHITECTURE AND DEPLOYMENT

The mule application has the following three flows which are routed through an APIKIT router:-

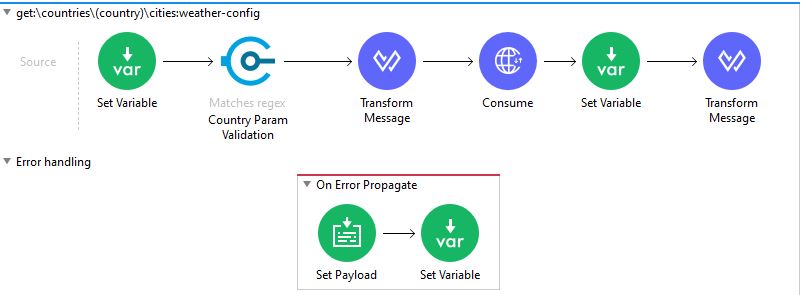
1. Get countries

URL: <http://localhost:8081/api/countries>



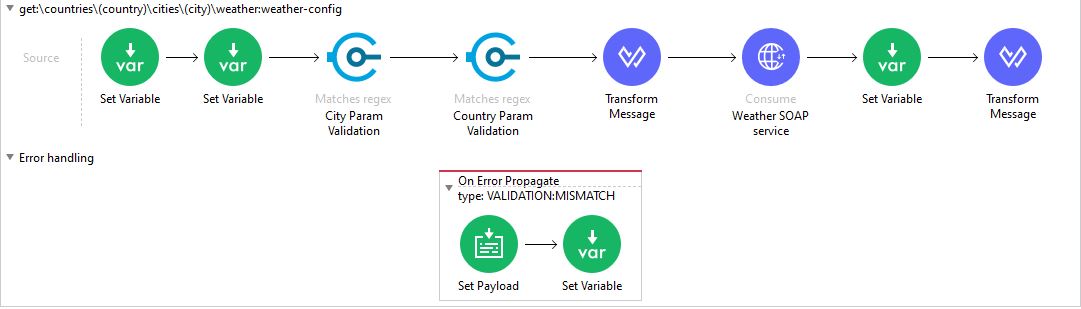
1. Get cities

URL: http://localhost:8081/api/countries/{country}/cities



1. Get weather

URL: [http://localhost:8081/api/countries/{country}/cities/{city}/weather](http://localhost:8081/api/countries/%7bcountry%7d/cities/%7bcity%7d/weather)



**Instructions to run the project:**

1. NODE

Clone the aforementioned repository using the following command.

git clone https://github.com/sachinarora92/weather-app-deloitte.git

Go to weatherExcerciseDockerFile folder and run npm install

Go to weatherExcerciseDockerFile folder and execute build the docker image and run it as a container or, simply run `node server.js` to start the Weather SOAP WS

Go to `http://localhost:8080/GlobalWeather?WSDL` and make sure it answers the WS spec

2. MULE EE RUNTIME [4.3.3]

Go to https://www.mulesoft.com/lp/dl/studio and download Mule Anypoint Studio 7 and Mule 4

Unzip the downloaded file to create the folder

3. MULE APPLICATION

Copy the JAR file `weather-adapter-rest.jar` into the app folder of the Mule Enterprise Standalone folder unzipped in the previous step

Be sure to have JRE or JDK 8 installed

Go to `/bin` Mule standalone subfolder and run mule.sh in Linux or mule.bat in Windows

The project code can also be imported into the Anypoint studio and can be executed.

4. API Routes

Open postman or simply go to a browser and navigate to these URLs:

http://localhost:8081/api/countries

http://localhost:8081/api/countries/Australia/cities

http://localhost:8081/api/countries/Australia/cities/Melbourne/weather

## RAML

#%RAML 1.0

---

title: Weather REST APIs

/countries:

get:

responses:

200:

body:

application/json:

example: |

{

"countries": [

{"country":"Australia"},

{"country":"Newzealand"}

]

}

/{country}:

#get:

/cities:

get:

responses:

200:

body:

application/json:

example: |

{

"cities" : [

{

"city":"Melbourne",

"country":"Australia"

},

{

"city":"Archerfield Aerodrome",

"country":"Australia"

}

]

}

/{city}:

#get:

#put:

#patch:

/weather:

get:

responses:

200:

body:

application/json:

example: |

{

"location":"Melbourne",

"Time":"11 AM",

"Wind":"15 km per hour",

"Visibility":"10 km",

"SkyConditions":"sunny",

"Temperature":"18",

"DewPoint":"2 C",

"RelativeHumidity":"35",

"Status":"Normal"

}

#post:

#put:

#delete:

The above RAML specifies the contract of the rest API.

## CHALLENGES

* The SOAP webservice at URL http://www.webservicex.com/globalweather.asmx?WSDL was unavailable. Used Docker image for the service.
* Webservice provide with the docker image embedded the content in two CDATA elements (<![CDATA[ <![CDATA[ ]] ]]) which was wrong. Corrected the docker image. New docker sample is available in repo.
* This is the first time I have used Mulesoft Anypoint for development. So some code or design decisions may be naive. The intent of this exercise is to learn a new technology like MuleSoft AnyPoint and to demonstrate an integration scenario using it. This code should not be used in production and should just be used as a reference.
* Anypoint studio was not able to provide the support for dataweave expression checker. I had to run the application again and again to check the expression’s result.

## BACKLOG and FUTURE SCOPE

* Security aspects are not included or demonstrated.
* Unit tests are added only for happy path scenarios.
* Error handling is done at the generic level. No detailed description is available in the API response. For unavailable cities/countries, it just returns 400 BAD request {“message”:”Bad request”}.
* Cloudhub deployment is not supported.
* Caching strategies are not incorporated in the API.