Weather Information Application

System Architecture Document

Version 1.0

[Note: The following template is provided for use with the MuleSoft development projects. Text enclosed in square brackets and displayed in blue italics (style=InfoBlue) is included to provide guidance to the author and should be deleted before publishing the document. A paragraph entered following this style will automatically be set to normal (style=Body Text).]

[To customize automatic fields in Microsoft Word (which display a gray background when selected), select File>Properties and replace the Title, Subject and Company fields with the appropriate information for this document. After closing the dialog, automatic fields may be updated throughout the document by selecting Edit>Select All (or Ctrl-A) and pressing F9, or simply click on the field and press F9. This must be done separately for Headers and Footers. Alt-F9 will toggle between displaying the field names and the field contents. See Word help for more information on working with fields.]

Revision History

|  |  |  |  |
| --- | --- | --- | --- |
| **Date** | **Version** | **Description** | **Author** |
| 04/10/2019 | 1.0 | Weather Information API | Harishankar Kaimal |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

[Introduction](#_kdeuoxbbrzc)

[1.1 Purpose](#_nlfs80ohxtfu)

[1.2 Scope](#_g6fbbg48ocse)

[1.3 Definitions, Acronyms, and Abbreviations](#_glape1ch94ib)

[1.4 References](#_xbn7coju2x3a)

[1.5 Overview](#_i69czmg510fv)

[1.6 Assumption](#_yomg3skotl37)

[2. Architectural Goals and Constraints](#_pv4x3rxomye2)

[2.1 Context](#_wopycba3k0s7)

[2.2 Current State](#_92xti5iiptie)

[2.3 Approach and Methodology](#_oezx0jpzu1zs)

[2.3.1 API Led Connectivity (Conceptual Approach)](#_g4o6m2irsjrm)

[2.3.1.1 System Layer](#_9yjhz9oqaqnv)

[2.3.1.2 Process Layer](#_tv5766n4k5lh)

[2.3.1.3 Experience Layer](#_hlvvgd7yrteq)

[2.3.2 Mapping with <CUSTOMER> Methodologies](#_7fz02rogx0c6)

[3. Architectural Representation](#_7rsvucn5hbqq)

[4. Use-Case View](#_w5eldp45o4ia)

[4.1 Use-Case Realizations](#_gq6xg9g8c9m0)

[5. Logical View](#_dgwd70kvrt04)

[5.1 Overview](#_c6uwr48khkiu)

[5.2 Architecturally Significant Logical Flows](#_6jsp8ggsco4i)

[6. Deployment View](#_vv8kzzgdnab5)

[7. Process View (optional)](#_vpl9wpm3jg93)

[8. Implementation View (optional)](#_qgid5cvipkd0)

[8.1 Overview](#_7ker61pttsq2)

[8.2 Layers](#_z9z6dsl1x19s)

[9. Data View (optional)](#_lcb4ryxpjwh5)

[10. Integration Patterns](#_wvbskiea26d6)

[11. Software Development LifeCycle](#_kf0hss9gcaak)

[11.1 APIs LifeCycle](#_l9ukv7ws9d7a)

[11.2 Development principles / guidelines](#_rbdyv9gfu8g8)

[12. Size and Performance](#_bupw63jchcw9)

[12. Non Functional Requirements and Service Level Agreements](#_u79t8q3o9v29)

[12.1 Failure Points Analysis](#_oy0z1tqiqomq)

[12.2 Mechanisms](#_tgfsjakqggej)

[12.2.1 Specific Mechanisms](#_4znt2istkdfa)

[12.2.1.1 Testing Strategies](#_ga7tjl7xatrw)

[12.2.1.1 Performance Testing Plan](#_muy7b08a9uj4)

[12.2.1.3 Performance Testing Tools](#_1r9be3knrcy6)

[12.2.1.4 Performance Testing load scenarios](#_j3ngs6oc5f48)

[12.2.1.5 Dashboard of interest in Iterations](#_od4moqe7u8g1)

[12.2.1.6 Alerts/Notifications](#_bac0sqtacnf)

[12.2.1.7 Logging](#_qa8h928991hd)

[12.2.1.8 Error Handling](#_ag1frqr8sd7k)

[12.2.1.8.1 Validation Strategy](#_i6srl1c5mzot)

[12.2.1.8.2 Error Handling Strategy](#_6az20blw1ak4)

[12.2.1.8.3 Flow Level Exception Handling](#_8e2m7zgf2mx4)

[12.2.1.8.4 Application Level Exception Handling](#_xorko0dczowo)

[12.2.1.8.5 Global Exception Handling](#_amb9mfltrn3i)

[12.2.1.8.6 APIKit Exception Handling](#_lbx5s3fevzic)

[12.2.1.8.7 HTTP Status Code](#_43kabhkciece)

[12.2.1.9 Failure recovery strategies](#_y7pyjwqizsgy)

[12.2.1.9.1 Reconnection Strategies for connectors](#_86vkbj143nuy)

[12.2.1.9.2 Retry strategies for flows](#_hb6nui5y3mie)

[12.2.1.9. 3 Redelivery strategies for flows](#_cv0w6nir64xv)

**System Integration Architecture Document**

# **Introduction**

This document is intended to serve as an initial reference Architecture for future and ongoing developments.

Thus, it should be treated as a living document and tailored over time with customer needs and experiences from each project.

The architecture is aimed to cover common use cases and provide a quick reference for each approach from an “REST design pattern” perspective and describe how developers can leverage MuleSoft platform capabilities to implement those patterns.

## **1.1 Purpose**

This document provides a comprehensive architectural overview of the system, using a number of different architectural views to depict different aspects of the system. It is intended to capture and convey the significant architectural decisions which have been made on the system.

## **1.2 Scope**

This document covers the design and development essentials of Weather information API product

## **1.3 Definitions, Acronyms, and Abbreviations**

[This subsection provides the definitions of all terms, acronyms, and abbreviations required to properly interpret the **System Integration Architecture Document**. This information may be provided by reference to the project’s Glossary.]

|  |  |
| --- | --- |
| **Term** | **Definition** |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |

## **1.4 References**

[This subsection provides a complete list of all documents referenced elsewhere in the **System Integration Architecture Document**. Identify each document by title, report number (if applicable), date, and publishing organization. Specify the sources from which the references can be obtained. This information may be provided by reference to an appendix or to another document.]

|  |  |  |
| --- | --- | --- |
| **Author** | **Document Name** | **Description** |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

## **1.5 Overview**

The following sections comprises of architectural goals and conceptual approaches adopted during the design application.

## **1.6 Assumption**

# **2. Architectural Goals and Constraints**

## **2.1 Context**

Expectations:

* Cloud/OnPrem/Hybrid Architecture based on customer requirement
* Setup environments (sandbox and production)
* Setup Identity Management
* Review of n use cases
* Core Framework definition
* C4E – Formation of c

## **2.2 Current State**

* <CUSTOMER experience developing with Mule.>
* <CUSTOMER subscription (OnPrem | CloudHub | Hybrid, API Management + Analytics?, VPC, etc)>
* <# of expected use cases>
* <Source code tools used, GitHub>
* <Documentation Tools used>
* <CI/CD Tools used>
* <Stakeholders and roles>
* <Mule version and tools to be used>

## **2.3 Approach and Methodology**

The proposed approach is based on the following methodologies/guidelines

* **Domain Driven Design (DDD)** as a design approach to understand the business problem/solution and to develop solid building blocks that compose the application network
  + Domain Events Brainstorming
  + Define Domains, Value Objects, Aggregates, etc
  + Define Bounded Contexts
  + Define Context Mappings
* **API-Led Connectivity** as a reference focused on:
  + Architectural layout, layered approach
  + Taking into account the speed of change
  + Including teams roles separation
* **<C4E** as a final goal to build a solid foundation with the team>

## **2.3.1 API Led Connectivity (Conceptual Approach)**

### 2.3.1.1 System Layer

System APIs will provide a means of accessing the underlying/core weather systems exposing that data in a canonical format, while providing downstream isolation from any interface changes or rationalization of those systems. These APIs will also change more infrequently and will be governed by Central IT given the importance of the underlying systems.

### 2.3.1.2 Process Layer

The underlying business processes that interact and shape this data should be strictly encapsulated independent of the source systems from which that data originates, as well as the target channels through which that data is to be delivered. These APIs perform specific business processes functions and provide access to non-central data and may be built by either Central IT or Line of Business IT.

### 2.3.1.3 Experience Layer

Data is now consumed across a broad set of channels/teams, each of which want access to the same data but in a variety of different forms. Experience APIs are the means by which data can be reconfigured so that it is most easily consumed by its intended audience, all from a common data source, rather than setting up separate point-to-point integrations for each channel.

The idea of applying this approach is to have scalable and reusable services, lower maintenance costs, faster time-to-market, flexibility, agility and a low learning curve.

## **2.3.2 Mapping with Methodologies**

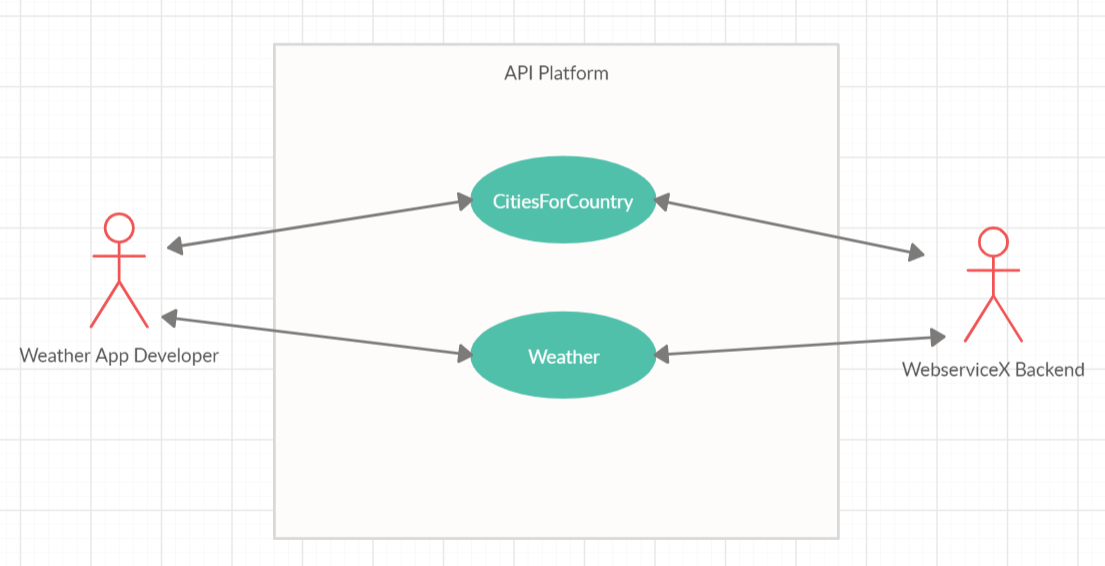
N/A

# **3. Architectural Representation**

# 

# **4. Use-Case View**

[This section lists use cases or scenarios from the use-case model if they represent some significant, central functionality of the final system, or if they have a large architectural coverage—they exercise many architectural elements or if they stress or illustrate a specific, delicate point of the architecture.]



## **4.1 Use-Case Realizations**

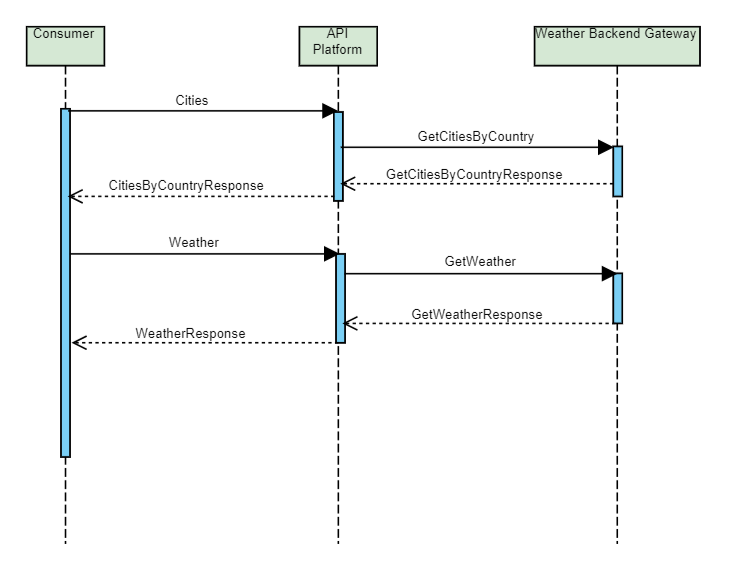
The system has a weather information functional requirement. The customer use cases involve interaction between the Customer Support Team (actor) and WebserviceX backend service.

# **5. Logical View**

## **5.1 Overview**

## **5.2 Architecturally Significant Logical Flows**

The information regarding cities by country is provided by CitiesByCountry API. The weather details for the respective city selected is provided by Weather API

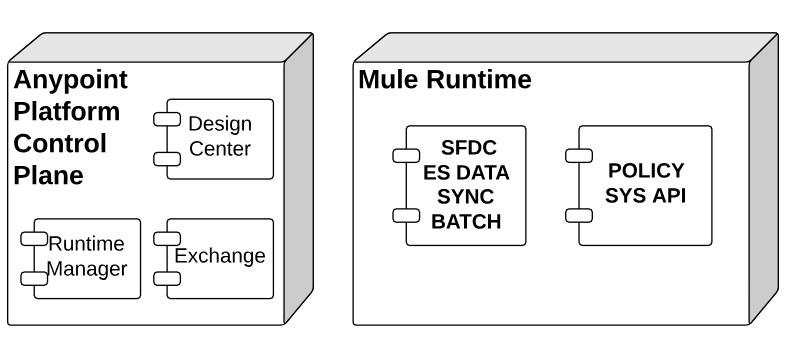


# **6. Deployment View**

The infrastructure is composed and divided by the following environments:

* Prod
* Sandbox

## Example deployment:



# **7. Process View (optional)**

[This section describes the system's decomposition into lightweight processes (single threads of control) and heavyweight processes (groupings of lightweight processes). Organize the section by groups of processes that communicate or interact. Describe the main modes of communication between processes, such as message passing, interrupts, and rendezvous.]

# **8. Implementation View (optional)**

[This section describes the overall structure of the implementation model, the decomposition of the software into layers and subsystems in the implementation model, and any architecturally significant components.]

## **8.1 Overview**

[This subsection names and defines the various layers and their contents, the rules that govern the inclusion to a given layer, and the boundaries between layers. Include a component diagram that shows the relations between layers. ]

## **8.2 Layers**

[For each layer, include a subsection with its name, an enumeration of the subsystems located in the layer, and a component diagram.]

# **9. Data View (optional)**

[A description of the persistent data storage perspective of the system. This section is optional if there is little or no persistent data, or the translation between the Design Model and the Data Model is trivial.]

# **10. Integration Patterns**

**Synchronous Integration Pattern**: The connection between the sender and replier (in the API context, this would be typically HTTP connection) stays open during this period of time. This type of communication is essential when the sender application needs an immediate response to continue with data processing

# **11. Software Development LifeCycle**

Source Code:

Staging

* Local Development
* Development environment
* Testing
* Intermediate stages (QA/UAT)
* Production

Testing

* Unit
* Integration
* Regression
* Performance
  + Load
  + Stress
  + Soak
* User Acceptance

## **11.1 APIs LifeCycle**

## **11.2 Development principles / guidelines**

* Mule Development Recommendations (if any)
* Naming Conventions in the project

# **12. Size and Performance**

# **12. Non Functional Requirements and Service Level Agreements**

The weather data should be reliably retrieved from the API. The load balancing or clustering solution can be considered for high availability and it has to be cost effective.

The response time for API should not be more than 300 milliseconds and throughput expected for API is 20 TPS.

Configuration management should keep all credentials secure.

## **12.1 Failure Points Analysis**

## **12.2 Mechanisms**

### 12.2.1 Specific Mechanisms

#### 12.2.1.1 Testing Strategies

#### 12.2.1.1 Performance Testing Plan

#### 12.2.1.3 Performance Testing Tools

#### 12.2.1.4 Performance Testing load scenarios

#### 12.2.1.5 Dashboard of interest in Iterations

#### 12.2.1.6 Alerts/Notifications

#### 12.2.1.7 Logging

An efficient logging mechanism has been implemented at the API pre and post invocation of backend proxy

#### 12.2.1.8 Error Handling

On error propagate scope has been used extensively to propagate the exceptions returned by API and backend proxy

##### 12.2.1.8.1 Validation Strategy

##### 12.2.1.8.2 Error Handling Strategy

##### 12.2.1.8.3 Flow Level Exception Handling

On error propagate scope has been effectively used to handle the flow level exceptions

##### 12.2.1.8.4 Application Level Exception Handling

##### 12.2.1.8.5 Global Exception Handling

##### 12.2.1.8.6 APIKit Exception Handling

Various exceptions returned by API KIT router post validation of the request has been handled using on error propagate scope

##### 12.2.1.8.7 HTTP Status Code

Please refer to the API Specification Document\_v1 for more details

#### 12.2.1.9 Failure recovery strategies

##### 12.2.1.9.1 Reconnection Strategies for connectors

Please refer to the API Specification Document\_v1 for more details

##### 12.2.1.9.2 Retry strategies for flows

##### 12.2.1.9. 3 Redelivery strategies for flows

# 