The design for modularization and one generic function for any rule

Contents

[Abstract 1](#_Toc110520691)

[Code configurability 1](#_Toc110520692)

[Function code refactor approach 1](#_Toc110520693)

# Abstract

This document covers the key aspects to refactor the POC’s under C4C-SBS-Extensions Atlas MongoDB app services. Major issues with the current code are for all three different POC’s we have specific functions created for update, get, patch and execute rule function. Whilst refactor the code the overwhelming number of functions should be able to reduce, and individual functions will be created specific to the POC’s. This document describes how the different functions can be standardized and make it specific to a feature. It also covers the configurability aspects to fetch values from a configuration value store available in MongoDB Atlas app services.

Asynchronous Rules Evaluation for Field Updates:

The Asynchronous Rules evaluation is based on the premise that the user perform actions which triggers events to further do updates to the C4C fields that need to be updated not instantaneously but can be scheduled, or get updated after the fact. And does not affect the downstream of the process immediately. This is a fire and forget Stateless approach for Rules evaluation to be used for asynchronous / deferred field updates .

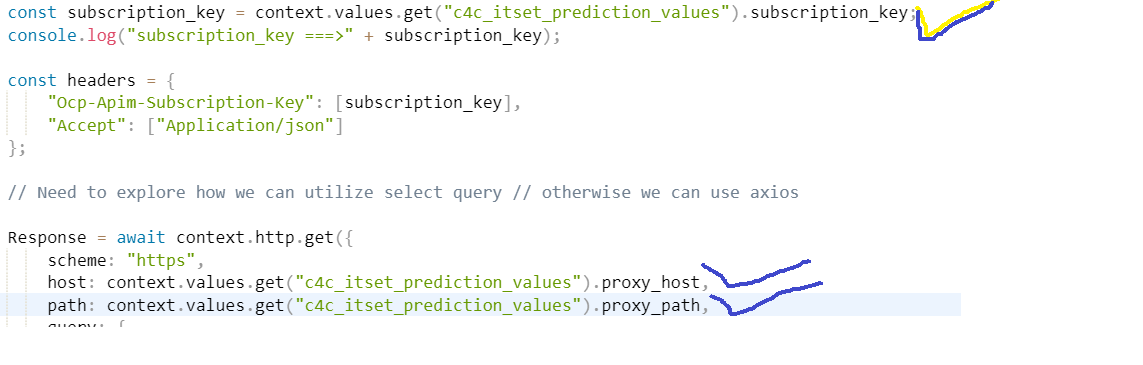
The pattern is suggested for processes which do not need stateful execution from the C4C stack and depends on the invocation of the webhook external to the stack sitting on cloud as shown in the side by side rules evaluation depicted below.

The C4C systems invokes external rules evaluation synchronously in a single Post to the Rules Evaluation engine , which also receives the Parameter ID's in the request Parameters as shown, which identified the Rule Sets to be evaluated and the data required for the evaluation. The Mapping between the Parameter like Ticket\_Sync\_update and the rules is maintained in the Rules Mapping collection.

# Code configurability

The code must be configurable. It should not contain any hardcoded values. All environment specific values must be stored in a configuration value store in Atlas MongoDB app services. Similarly, the configuration that can be stored in database also fetched and used. The secrets also must be stored and used inside the code, but it can be fetched from environmental specific secrets.

Few examples are shown below.



Design for modularization and one generic function for any rule

There will be only two functions are required. The first function will be invoked by c4c directly and the c4c event will be passed and that will be stored directly into the MongoDB. On the back of the event creation a database trigger will call a generic function which will execute any rule based on the fields intended value.

**The function which inserts the c4c Ticket event.**

This function just intakes the c4c event in the form the request body. This event will be formed here as JSON body and inserted into MONGODB collection.

const newItem = {

            "eventId": (new Date()).getTime(),

            "type": eventData.type,

            "id": eventData.id,

            "isReprocessed" : false,

            "ruleName" : ruleName,

            "eventPayload": eventData.data

        };

Naturally, the database trigger must be created upon every Save operation and full Document must be send to the generic function.

**One Generic function which takes care of Get/Evaluate Rule(s)/Patch operation**

* This will be the only function which takes care of get the ticket record from c4c APIM endpoint and execute the rule for the data availed and patch the delta changes back to c4c for a specific column depending upon any specific rule execution or a list of rules.
* The anonymous main function will receive the event data along with rule Name which determines a specific/all rule execution.
* Use configuration values as well as database configuration options to get environmental specific variables/constants to use inside code. The subscription, base URL, authorizationKey database cluster name, query etc configurable items.
* Include meaningful logging message and log it for the request data comes in.
* The **GetTicket** method performs the context.http.get call to APIM get endpoint using ticket\_root\_entityIdIn value.
* **EvaluateEvent** method does the rule engine execution against the input and respond with rule match found status for a list of rule executions.
* The input data to execute the rule engine will be created as per the below snippet.

let inputData = {};

if (data && data.attributes && data.attributes.length > 0) {

data.attributes.map((element) => {

if (element.name === 'CountryText') {

const ServiceRequestUsedAddress = request['ServiceRequestUsedAddress'];

console.log("ServiceRequestUsedAddress", JSON.stringify(ServiceRequestUsedAddress));

inputData[element.name] = ServiceRequestUsedAddress[element.name] ? ServiceRequestUsedAddress[element.name] : '';

} else {

inputData[element.name] = request[element.name] ? request[element.name] : '';

}

console.log("inputData", element.name, request[element.name]);

});

}

* In the **PatchTicket** method, Identify the set of fields which need an update in the ticket record, form the Patch body and execute the patch method using context.http.patch depending on the data to be patched for a POC when the rule match found.
* Once the patch operation succeeded make sure log the status based on the status of patch operation completed

|  |  |  |  |
| --- | --- | --- | --- |
| **Function Name** | **Parameters** | **Route** | **Psuedocode** |
| **C4C\_TICKET\_EVENTS\_PRODUCER\_HTTP** | request, response | **/ C4C\_TICKET\_EVENTS\_PRODUCER\_HTTP** | This function just intakes the c4c event in the form the request body. This event will be formed here as JSON body and inserted into MONGODB collection.  A Database trigger will be created which will fire upon every event creation and be configured to hit the below generic function mentioned. |
| **C4C\_TICKET\_UPDATE\_DB** | Request, Response | /**C4C\_TICKET\_UPDATE\_DB** | •This will be the only function which takes care of get the ticket record from c4c APIM endpoint and execute the rule for the data availed and patch the delta changes back to c4c for a specific column depending upon any specific rule execution or a list of rules.  •The anonymous main function will receive the event data along with rule Name which determines a specific/all rule execution.  •Use configuration values as well as database configuration options to get environmental specific variables/constants to use inside code. The subscription, base URL, authorizationKey database cluster name, query etc configurable items.  •Include meaningful logging message and log it for the request data comes in.  •The GetTicket method performs the context.http.get call to APIM get endpoint using ticket\_root\_entityIdIn value.  •EvaluateEvent method does the rule engine execution against the input and respond with rule match found status for a list of rule executions.  •The input data to execute the rule engine will be created as per the below snippet.  let inputData = {};  if (data && data.attributes && data.attributes.length > 0) { data.attributes.map((element) => {  if (element.name === 'CountryText') {  const ServiceRequestUsedAddress = request['ServiceRequestUsedAddress'];  console.log("ServiceRequestUsedAddress", JSON.stringify(ServiceRequestUsedAddress));  inputData[element.name] = ServiceRequestUsedAddress[element.name] ? ServiceRequestUsedAddress[element.name] : '';  } else {  inputData[element.name] = request[element.name] ? request[element.name] : '';  }  console.log("inputData", element.name, request[element.name]);  });  }  •In the **PatchTicket** method, Identify the set of fields which need an update in the ticket record, form the Patch body and execute the patch method using context.http.patch depending on the data to be patched for a POC when the rule match found.  •Once the patch operation succeeded make sure log the status based on the status of patch operation completed. |