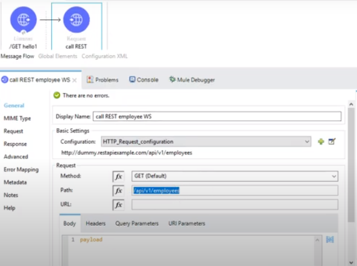
**1) Types of Integration:**

* File based Integration pattern
* JDBC Integration pattern
* Batch Processing and scheduled based Integration pattern
* Publish Subscribe pattern
* Polling Consumer pattern
* Point to Point Integration pattern
* Guaranteed Delivery message pattern
* Operating system Command Execution pattern
* Webservice Synchronous Consumer pattern
* Webservice Provider Pass-through pattern

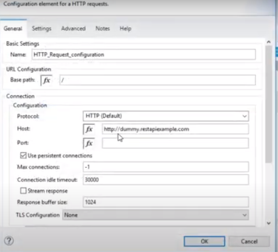
**2) Components for REST vs SOAP**

* For REST API, use HTTP 'Request'.
* For SOAP API, use 'Web Service Consumer' component.

**For REST API, Below is the flow**



Edit the connection wizard like below with the REST API url:



**3) Scheduler Component**

For a Scheduler based event trigger, use 'Scheduler' component in a Flow so that the scheduler will get triggered based on the CRON time that we mentioned and it can trigger other components/Flows

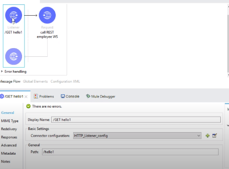
* Under the 'Scheduling Strategy' , set the frequency as either 'Fixed Frequency' or 'CRON' and other parameters like Daily, Hourly etc

**4) To Create SOAP API**

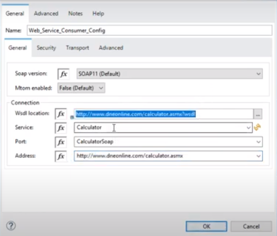
Under the 'Add Modules’, Add 'Web Service Consumer' component by dragging and dropping it. Now use the consume method inside it and configure it.

Now once this is added into the Module, The relevant Jar files will be added inside the project.

* Add the HTTP listener to invoke the consumer
* Configure the wsdl parameter to point the wsdl url from where you want to consume the data like below



Below are the wsdl information from the sample example from website:



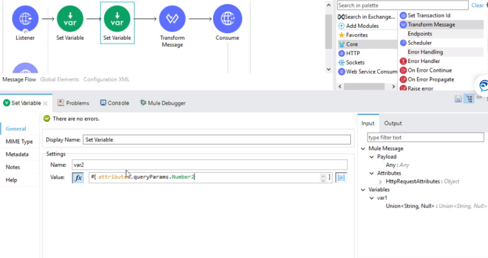
Now, we can pass the numbers from the postman rather than hardcoded in ‘set variable’

We are passing the value parameters from postman:

Add ‘Transform Message’ component to map the input variables to the output variables by mapping the variables in the mapping area:

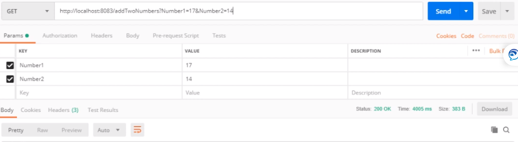
In the ‘Set Variable’ settings value, do below:

fx #[attributes.queryParams.Number1



The parameters are passed as:

http://localhost:port#/addTwoNumbers/Number1=4&Number2=5



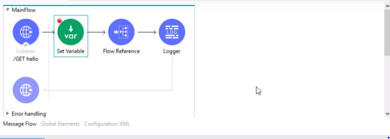
In order to get the output in the ‘Logger’ component, give below expression:

Message: fx : #[payload.body.AddResponse.AddResult

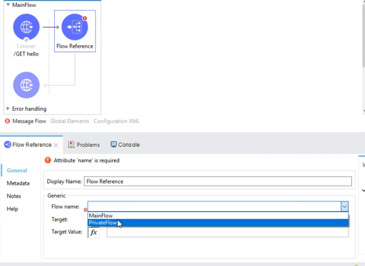
**5) Main Flow, Private Flow and Subflow :**

Here we can create a Main flow and then we can use a ‘Flow Reference’ component inside that to reference to either a **Private Flow** (it’s the same Flow but just renamed as Private Flow) or a ‘Subflow’ component’.

* Subflow are mainly a proven working model and can be reused/referenced in any flow, It will not have any error handler and the main flow’s error handler will be handling it.
* Based on conditions we can either invoke a Private Flow or a Subflow.



We can call the private flow or the Subflow from the ‘Flow Reference’s ‘Flow Name’ drop down list.



**6) Publish and Subscribe through VMConnector:**

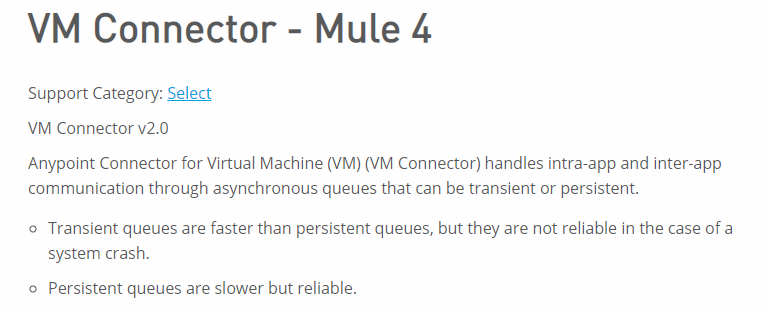
This Integration is used when you have to invoke from one mule app to another or to distribute one message to multiple recipients, we can use publish method and then the clients/consumers can subscribe to it to get the messages.

Refer to below Mule documentation for this:

<https://docs.mulesoft.com/connectors/vm/vm-connector>

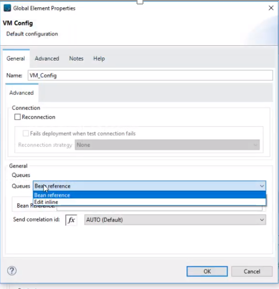
The messages can kept as Transient or persistent depending up the requirement.

This works in Asynchronous way.

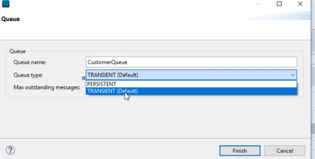


Below we can change the config properties to point ot ‘Edit Inline’ for Queues.

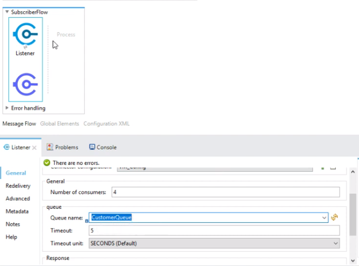
This will popup the QueueName and Type wizard and choose either ‘Persistent’ or ‘Transient’



Choose the Queue type:



Now let’s create subscriber Flow:



* Publish and Subscribe will happen in an Asynchronous way.
* PublishConsume will happen in a Synchronous way

Below we changed publish to publishConsume and have this consumed in our Subscribe Flow so that the control from publish flow will end only after it is consumed and thus happens in a Synchronous way.



**7) Scatter\_Gather Flow:**

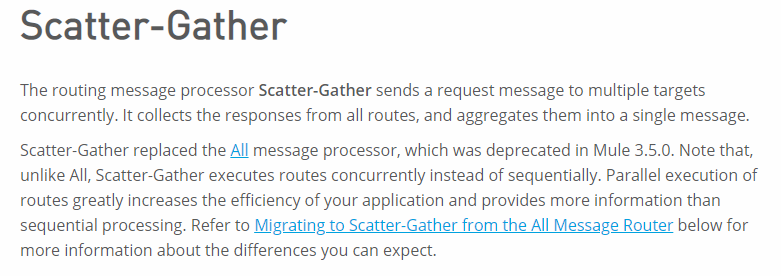
**Business Use case: Suppose you have data coming from 3 different systems like product from one system, price from another system and inventory from another source, they can all be fetched in parallel by using scatter gather, let’s say if each takes 3 seconds then total fetch time is still 3 seconds instead of 9 seconds if they run in sequence. So the main purpose is to increase speed and performance.**

**Below is the Link for Mule Documentation:**

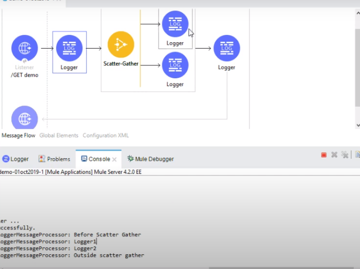
<https://docs.mulesoft.com/mule-runtime/3.9/scatter-gather>

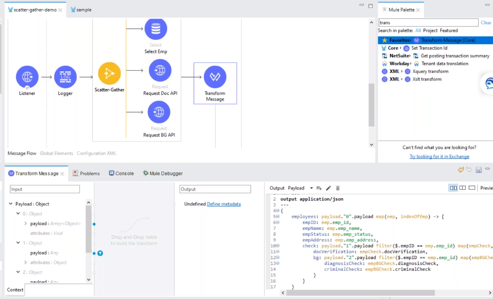
Scatter and Gather should have at least more than one flow coming out of it.

**What is Scatter-Gather.**



**Below it shows how the logger has been invoked showing the flow of Scatter Gather**





Above there is a map function which will loop through the data in the payload and a filter criteria used to filter and combine the 3 data from the scatter gather using the transform message.

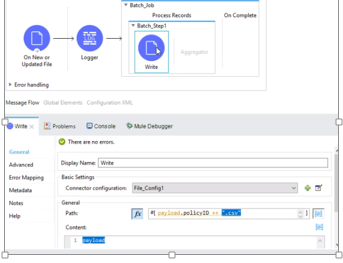
**8) BATCH PROCESSING (File processing):**

Mulesoft can do ETL like work by looking out for a file and then process them in a batch process for a huge file.

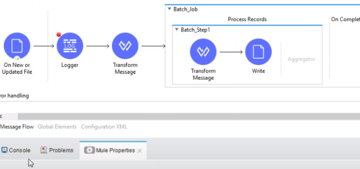
‘BatchJob’ component and ‘File write’ components can be used to read a bulk file and process it and output into an output file.

A Batch steps process 1600 records at a time with 100 records processed/thread in parallel at a time.

So total is 16 threads.



Below file is transformed from csv to Java using ‘Transform message’ and then again converted to java to csv after processing and before writing it into output file



**9) Invoke Static component:**

A file can be split up into multiple files by using a Java class and that java class can be invoked by ‘Invoke Static’ component

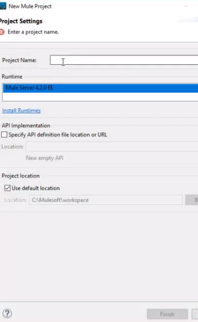
**10) How to import a SOAP wsdl into Mule Project**

Below click the option of ‘Specify API definition file location or URL’ so that we can import the wsdl.

The Mule project will be created with the SOAP components needed for the project.

We can create a wsdl using ‘Java Eclipse’ or any other Online SOAP development kit.

Then we can test the SOAP based mule project through the SOAPUI online otol.

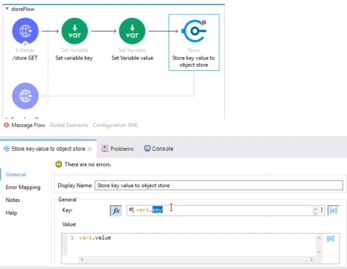




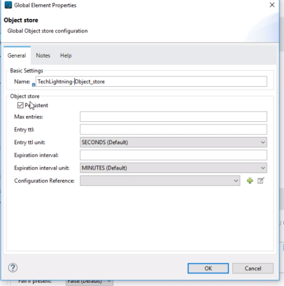
We can see where the SOAP API is imported into Mule project.

**11) ObjectStore:**

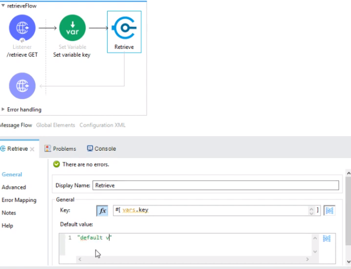
If we want to store and retrieve the parameters which have been passed on to the API end point then we can use ‘object store’ and ‘retrieve connecters to store the data



We need to choose ‘persistent’ option so that the value is stored on Disk.



Below you can retrieve the data using the retrieve component



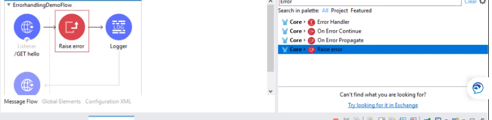
**12) Error Handling:**

There are 4 types of errors in Mulesoft

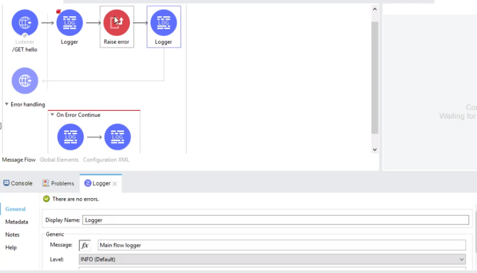
* Error Handler
* On Error Continue
* On Error Propagate
* Raise Error

Difference between ‘On Error Propagate’ and ‘On Error Continue’

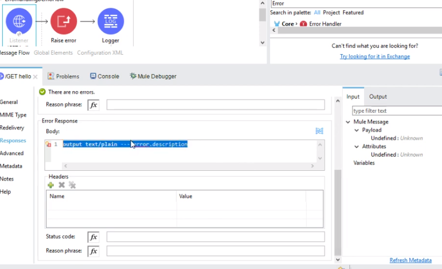
* On Error Continue will be exiting with Success
* On Error Propagate will be exiting with Failure



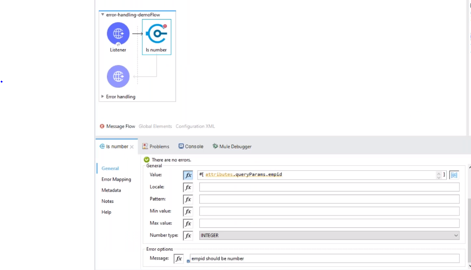
Below we can see how On Error Continue is placed inside the Error Handling component.



We can see below an Error response in displayed as Http listener response is configured with default error response.



**Example of Error handling:**



Above we can see if the queryparam parameter came in as a number or not/ if it is not a number then you can have global error handler to give On Error Propagate.

**Difference between ‘On Error Continue’ and ‘On Error Propagate’**

Example : Suppose if you want to look at an employee in a DB and if it is not there then you can do a ‘Try Cache Error handling for that DB check’ and then you can still continue to invoke other webservices to see if the employee is found on the Webservice call system or not. In that case you will use ‘on error continue’

**‘Raise Error’** is used whenever we want to throw a custom error message

**Example :** If the record is not found in the database then we can do a custom message instead of a regular message by using ‘Raise Error’ logic.

**‘Error Handling’** : Referencing the errors in a Global error handler across the flows is called ‘Error Handling’ so that all the errors are handled in one place.

**13) Mule Domain Project:**

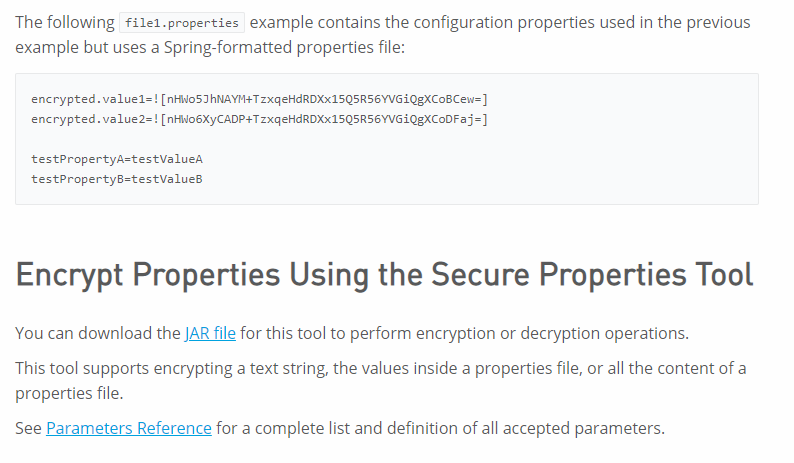
A Mule domain project is created for Mule run time applications to share their resources.

Like all the configurations from global components can be stored here and individual mule application’s properties can be mapped to this mule domain project so that the confogurations can be shared/reused by all the applications pointed to this ‘Mule Domain project’.

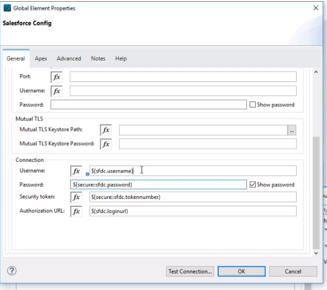
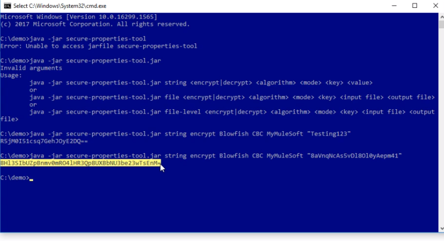
**14) Secure Configuration:**

Download the Jar file from below link and encrypt the credentials using this jar file and create the encrypted token and use that in connection properties for the component.

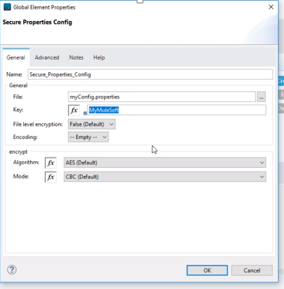
<https://docs.mulesoft.com/mule-runtime/4.2/secure-configuration-properties>



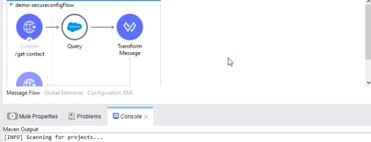
Below is how we encrypt using CLI



Now add the secure properties config from the Exchange into Global elements file and then point this secure config file like below:

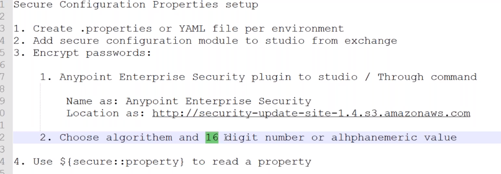


Below is a typical flow how we configured to connect to salesforce database using query method and by using seco=ure config parameters:



We can also secure the .yml and the passwords inside it by following the below steps using

‘Anypoint Enterprise Security plugin’



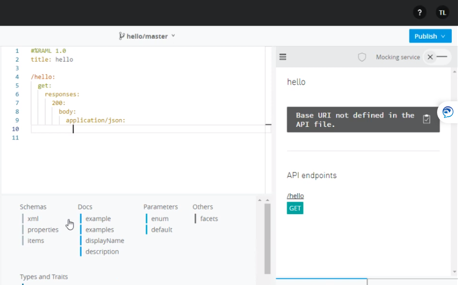
**15) Anypoint.mulesoft.com:**

Using Anypoint Cloud Portal we can do the following:

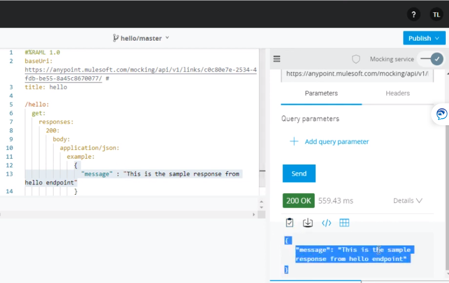
* We can create RAML in Design Center
* Publish RAML in Exchange
* Download RAML into Any point Studio
* Rn the Application
* Create API in API Manager
* Create a Policy
* Run the application

From the Design center, we can create a ‘New’ ‘API Specification’ to create an API.

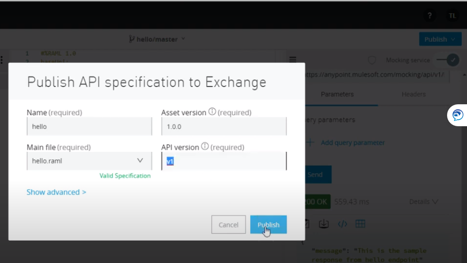
Create a RAML in API Specification.



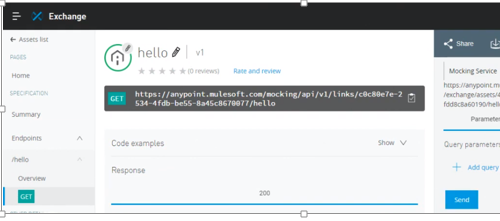
We can test it using Mock services, test url like below:



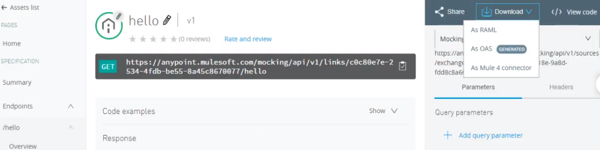
Once tested and everything looks Ok we can go ahead and publish it to Anypoint Exchange.



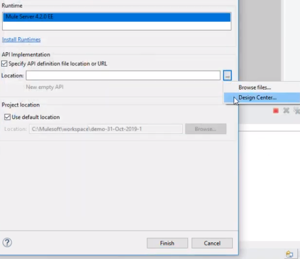
Now we can see the same in the exchange and start using it.



We can download it as RAML by downloading it from Exchange and use it your local as well:

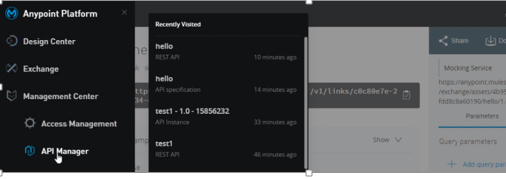


Now we can consume this RAML file from either using the local path where we stored the RAML File or form the design center like below:

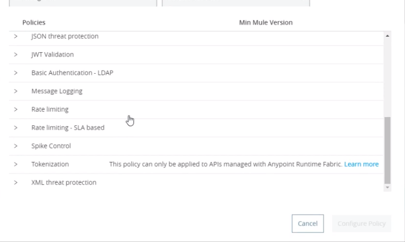


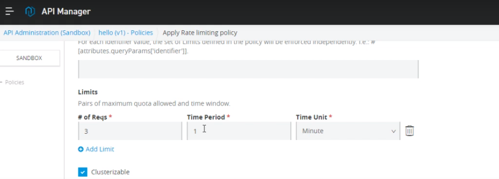
Now we can run the application in Anypoint studio.

Now we can manage this in API Manager in ‘Anypoint Platform’

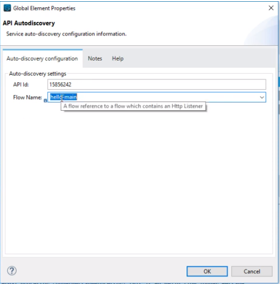


Now we can manage our API like setting the API Policy by setting the rate limit etc.





Now go the ‘Global elements’ in your Mule runtime and configure the apiid from the apimanager to the mule run tike apirouterkit



Through API Auto-Discovery method we can locate our API from the API Manager and then can link it to our runtime Mule application.

Through API Manager we can do below 2 things:

* Design Governance 🡪 like setting resources,methods etc
* Runtime Governance 🡪 like setting and managing policies like rats,ip blacklisting,white listing etc

**16) Mulesoft Choice Router:**

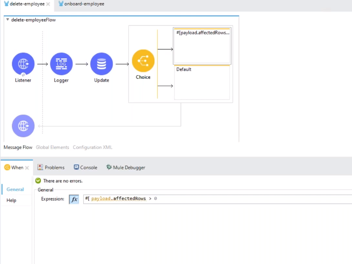
Refer to below link:

<https://docs.mulesoft.com/mule-runtime/4.2/choice-router-concept>

Based on the content in the payload we can make a conditional routing through ‘Choice Router’ which can call a specific flow based on the condition defined on the choice Router else the default choice flow will be used.

Example : Below we will invoke a HTTP Request body only if records are found in the DB else do not invoke the service which will save time and cost.

This is how all orchestrations needs to be done by thinking through the flow.



**17) Mulesoft Cluster setup:**

Download Mulesoft runtime client from below link:

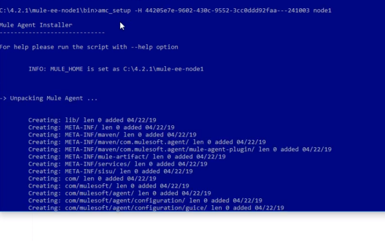
<https://www.mulesoft.com/lp/dl/mule-esb-enterprise>

You can copy the same and keep them as 2 separate nodes for a cluster.

Go to Anypoint Studio’s Runtime Manager and click Add servers.

Once you copied the agentid from add new server, go the node1 bin in server 1 and run the agent and do the same in the node2 as well.

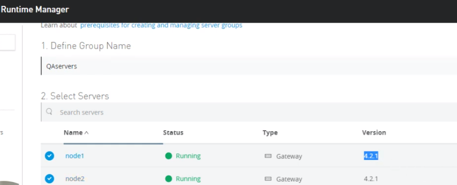
Invoke the mule in the 2 servers with 2 diff port#



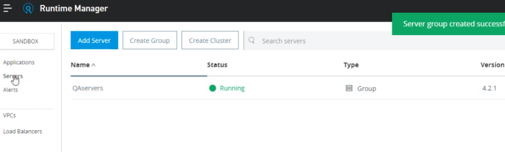
After Agent is installed, invoke Mule in that server node1



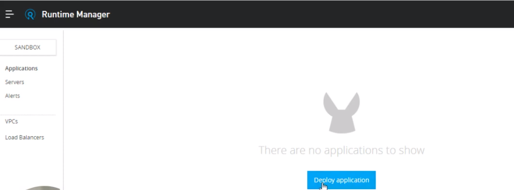
Now we can see the node1 and node2 are in running status.



Now we can see sever is running, earlier it was not running.

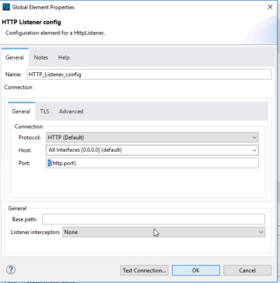


Now let’s deploy our application in this server:



Now we can create a mule project in our anypoint studio and then we can deploy that project’s jar file in this cluster:

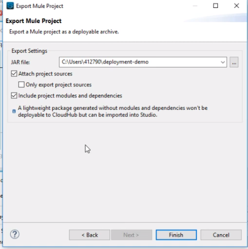
Instead of hardcoding the port#, we need to pull it from the config file.



By giving port# variable above

Now let’s export the mule project as a jar file by right clicking the mule project and choose export.

Now deploy this in the mule Runtimemnagaer inside the Qaservers.



Now instead of Groups, you can deploy the same in Cluster as well.

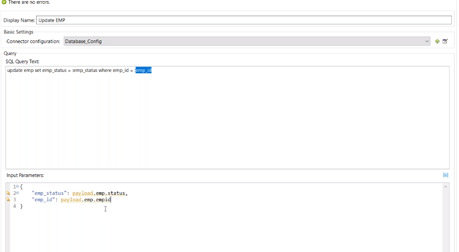
Click Create cluster, then add the node1 and node2 inside that and it will create that new Cluster.

**17) Database operations:**

UPDATE operation:

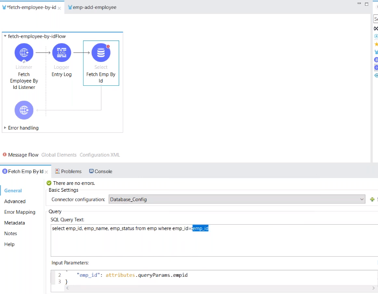
Below is the syntax to update dat ain table from the incoming updated data from POST oprtation

Where we map the payload data to the database columns.



**A SELECT statement from Query params.**

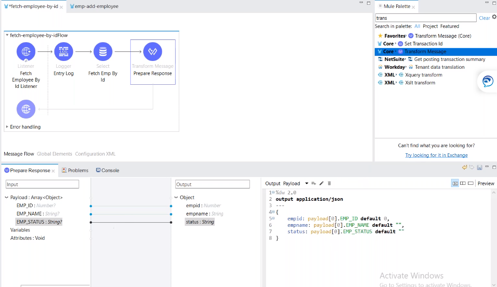
Below we can fetch the qyery param fetch id and fire it towards the database using attributes.queryparams dataweave language.



**Preparing the response using transform message:**

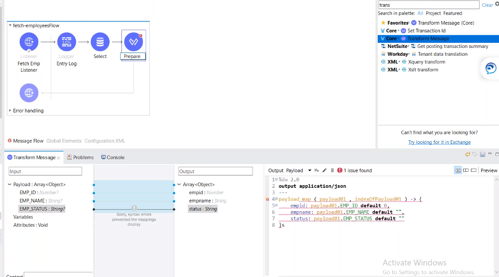
Below we shall prepare the response as a Json data by mapping the input data to the output data

Using dataweave metadata mapping.



**Below is for an aray of records which comes in the input:**

Here we use map function to parse(loop) through the incoming data array.

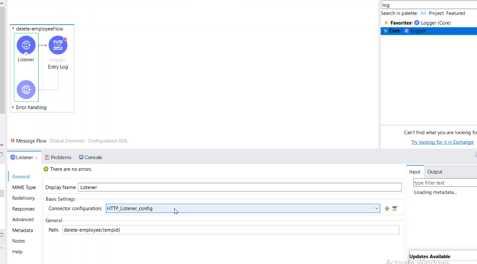


**How to map a key from a uri input where there is no key value**

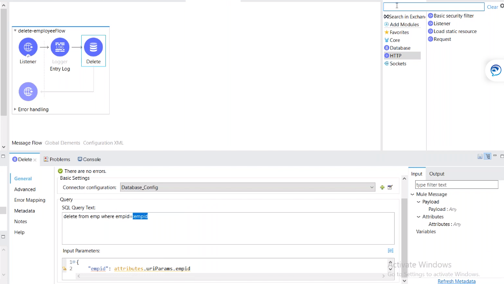
Now we should map/assign a key value for the incoming value, pelase see below where we ahev assigned {empid}



Below we are putting {empid} after the path to read the uri parameter.



**Below we are doing a Delete operation:**



**18) JMS (Java Messaging service):**

JMS works in below 2 ways:

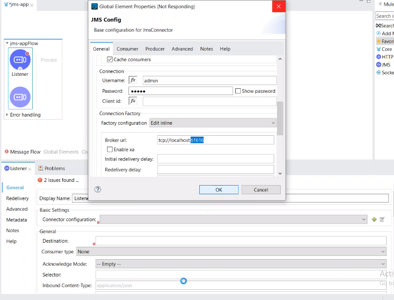
**Queue:** One to One ; Like a message can be published in a queue and can be consumed from a consumer from that queue

**Topic:** One to Many ; Like a message can be published to a topic and can be consumed by whoever subscribed to that topic.

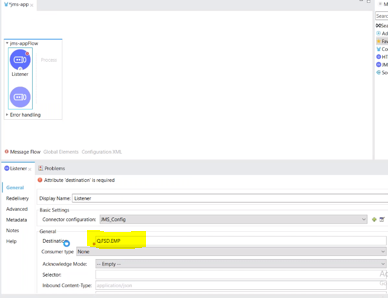
Based on requirements, we shall pick whether we need to orchestrate Queue or Topic and we can use JMS servers like ActiveMQ or RabbitMQ.

Configure the JMS listener:

Give the url of the ActiveMQ and its details.

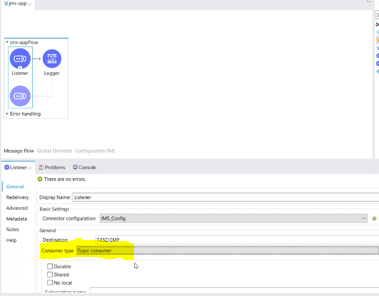


Now subscribe to the particular queue which is created in the ActiveMQ like below:



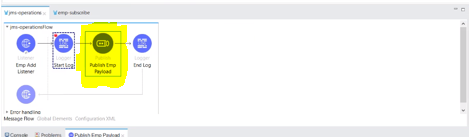
Now if we send any message in the queue, this JMS listener will keep listening to this queue and will fetch it .

Now if we want to subscribe to a Topic, then change the queue\_name to Topic\_name and change the

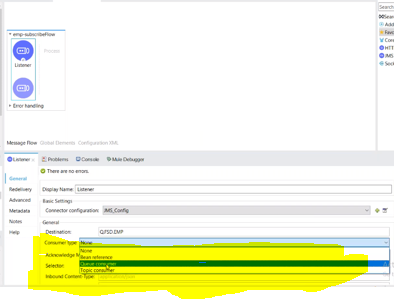


Now let’s publish the message in to a queue by using JMS Publisher:

Below we can connect the publisher with the ActiveMQ that we have.

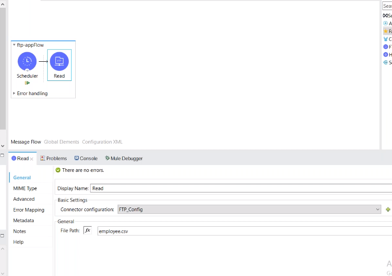


Now Consume the Queue or Topic by using a JMS Listener.



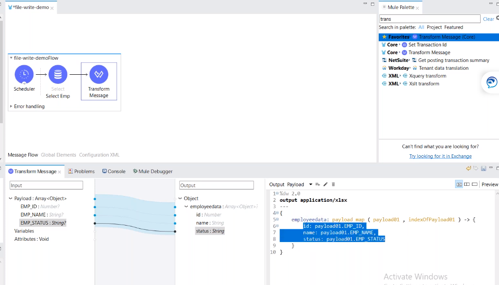
**19) READ file from STP server:**

We can use FTP component to read a file from ftp server and then read it based on a scheduler time usig Read component.



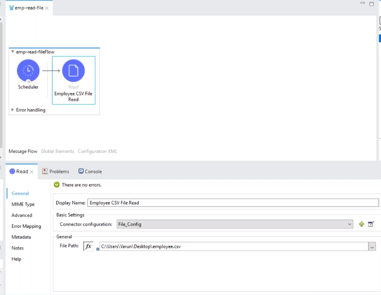
**20) Read Data from Database and write it to a csv file:**

Below we are reading data from database and writing it to a csv file by creating a datweave mapping by configuring the metadata like below.

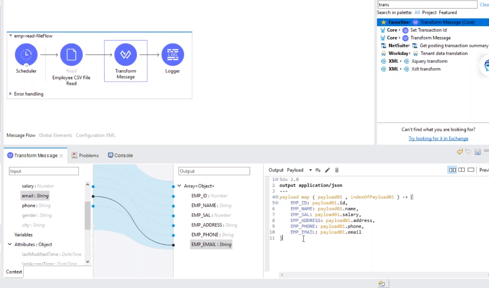


**21) Read a file from a path and load it in database.**

Now read the file from a path using a scheduler pull like below and configure like below from where to get the file.

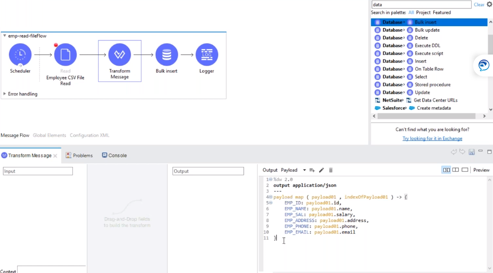


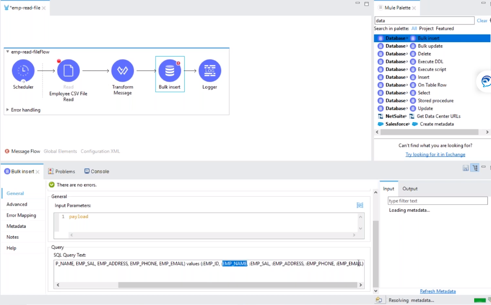
Let’s map the metadata like below to a json format.



Now let’s use **‘Bulk Insert’** component to load this bulk data into the database.

Below we have mapped the payload using the **‘Transform message’** and below is the metadata mapping

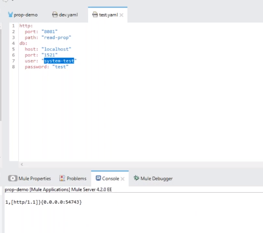




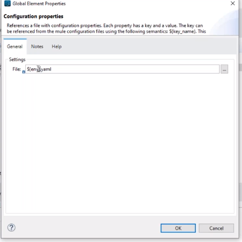
**21) Environment and Configuration file:**

We can run the same Mule Project in different enviornments by configuring its parameters to point to configuration file which is a .yaml file so that when we move to different enviornments we can just point the job to the correct environment and correct yaml file so that we don’t have to do any physical modifications.

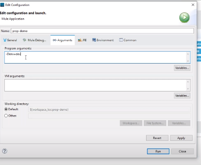
A sample .yml file.



Right click the project and go to ‘Run as’ and go to ‘Mule Application configure’ and then edit the Arguments as Dev environment like below:



Edit the config.yml file to ${env}.yml so that we can configure the environment like below.



**22) RAML (Restful API Modeling language):**

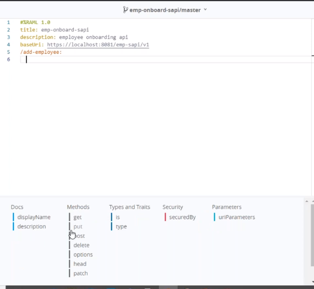
It’s a standard Mulesoft has developed to write a Restful service API in a specific format so that we can design, deploy, secure and manage the API.

Now what does an API as?

A method like POST,GET

* A Base URL
* Endpoint Path/Resource
* Body
* Response

Example below from API ‘Design Center’ where it suggests the options under each API component.with the help of that we can build out API.

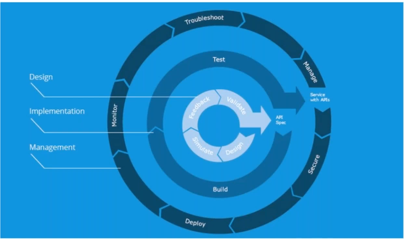


The inundation is very important just like writing a .yaml file.

RAML API design lifecycle:

Below is the cycle that we follow in RAML design.

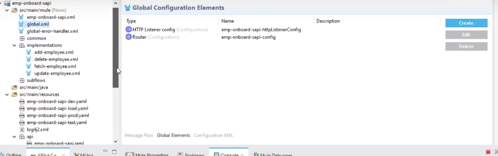
We can use **‘Mocking service’** to test the API that we develop using **‘Design Center’.**



**23) Organizing the project:**

We can organize the Mule project by putting them in folders like below so that it will look organized and easily readable.

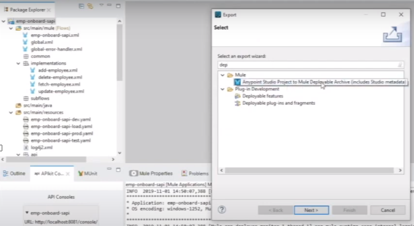
Like Global folder, implementations folder,subflow folder etc.

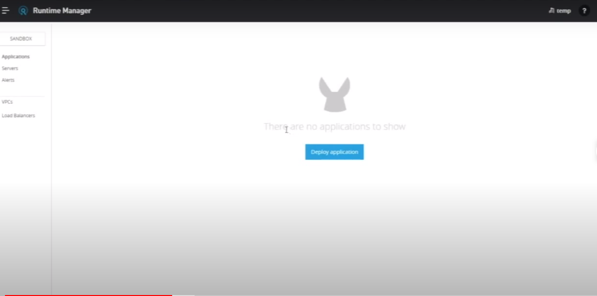


**24) Deploying an API or Mule Project to cloudhub:**

One approach is to export the Mule project to a Jar file and then import it into ‘Runtime Manager’ using ‘Deploy Application’.

Below wizard will generate a Jar file



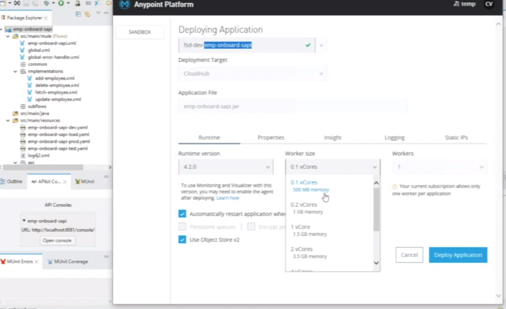


This approach will not be used as community edition allows only one free deployment through this way.

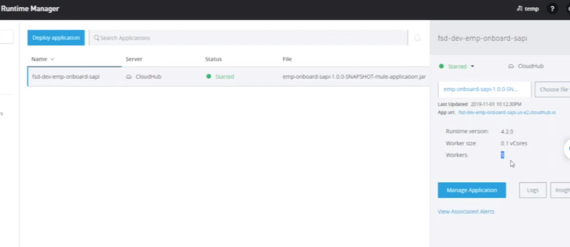
**2nd approach:**

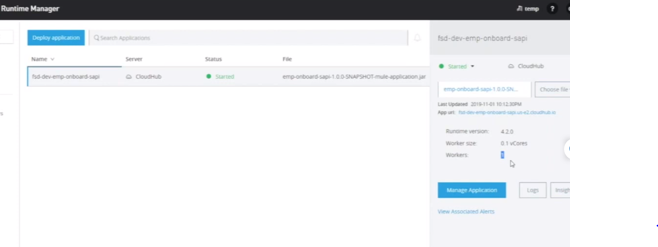
We can right click on the project and then click deploy to ‘cloudhub’

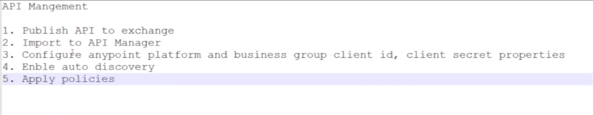
Below wizard will guide us through the naming conventions and settings.



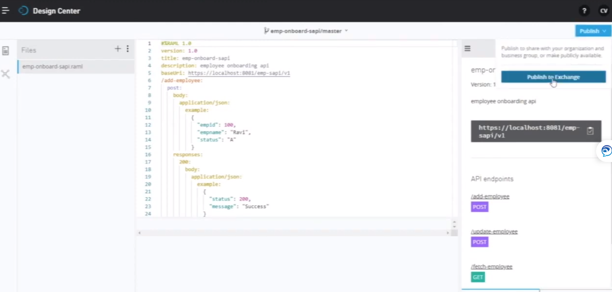
The application is now started and running in and we an see in the Runtime manager:



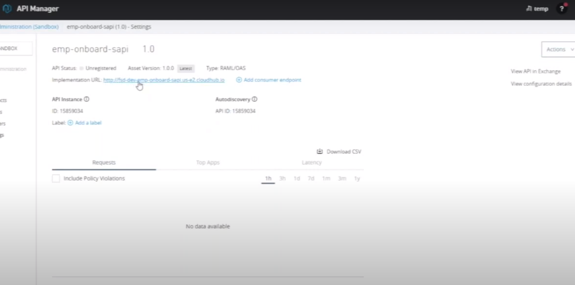




Now we can publish the API to the exchange

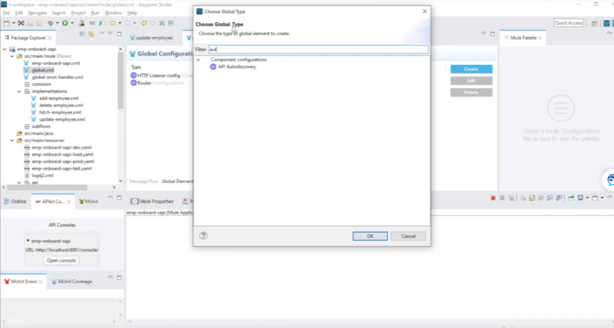


Now we need to link our cloudhub through Autodiscovery.

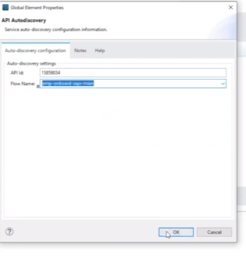


Take a copy of this API ID and then go to Anypoint studio and in global.xml

Enable autodiscovery

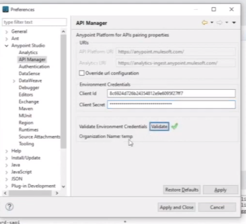


Enter the APIID here:



Now go to ‘Windows’ 🡪 ‘preferences’ ,under ‘Anypoint studio’ enter the ‘clientid’ and ‘client secret’ with the details that we get from ‘Access Management’

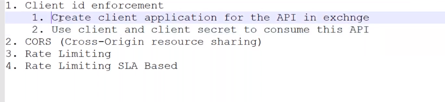
Also, validate it.



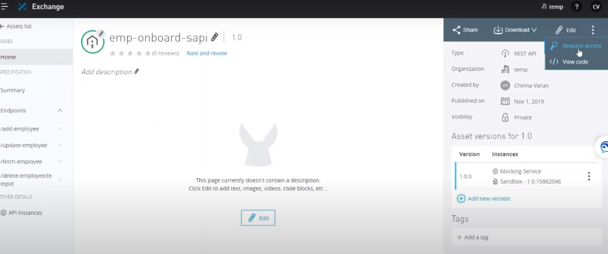
Now we need to deploy the application again by deploying it to cloudhub, once deployed, the API will become Active and get started.

**25) Policy Management:**

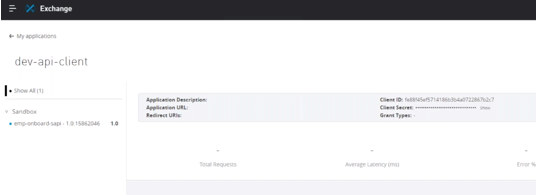
Now we can set some policies in the API and manage it.



Click on the API in the exchange and click ‘Request Access’ in order to generate the clientid and client secret for the application client like ‘pstman’ from which if we pass these values then only we can access this API else it will throw error.



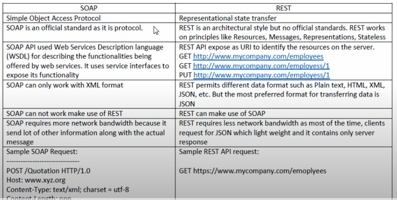
Now it will give us the clientid and client secret:

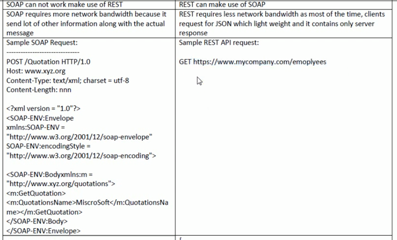


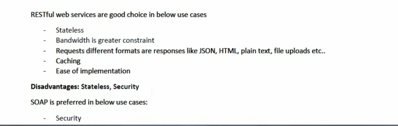
Now we need to apply the policy to this API

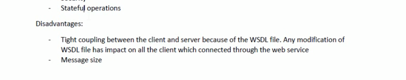
**26) SOAP Vs REST**

The Business Use case,Constraints,Requirements,security,network bandwidth and environment decides whether we need to use SOAP or REST as the Webservice









**27) HTTP Response odes:**

Below are different HTTP response codes that we can specify in an API.

|  |  |
| --- | --- |
| **S.N.** | **Code and Description** |
| 1 | **1xx: Informational**  It means the request has been received and the process is continuing. |
| 2 | **2xx: Success**  It means the action was successfully received, understood, and accepted. |
| 3 | **3xx: Redirection**  It means further action must be taken in order to complete the request. |
| 4 | **4xx: Client Error**  It means the request contains incorrect syntax or cannot be fulfilled. |
| 5 | **5xx: Server Error**  It means the server failed to fulfill an apparently valid request. |

HTTP status codes are extensible and HTTP applications are not required to understand the meaning of all the registered status codes. Given below is a list of all the status codes.

1xx: Information

|  |  |
| --- | --- |
| **Message** | **Description** |
| 100 Continue | Only a part of the request has been received by the server, but as long as it has not been rejected, the client should continue with the request. |
| 101 Switching Protocols | The server switches protocol. |

2xx: Successful

|  |  |
| --- | --- |
| **Message** | **Description** |
| 200 OK | The request is OK. |
| 201 Created | The request is complete, and a new resource is created . |
| 202 Accepted | The request is accepted for processing, but the processing is not complete. |
| 203 Non-authoritative Information | The information in the entity header is from a local or third-party copy, not from the original server. |
| 204 No Content | A status code and a header are given in the response, but there is no entity-body in the reply. |
| 205 Reset Content | The browser should clear the form used for this transaction for additional input. |
| 206 Partial Content | The server is returning partial data of the size requested. Used in response to a request specifying a *Range* header. The server must specify the range included in the response with the *Content-Range* header. |

3xx: Redirection

|  |  |
| --- | --- |
| **Message** | **Description** |
| 300 Multiple Choices | A link list. The user can select a link and go to that location. Maximum five addresses  . |
| 301 Moved Permanently | The requested page has moved to a new url . |
| 302 Found | The requested page has moved temporarily to a new url . |
| 303 See Other | The requested page can be found under a different url . |
| 304 Not Modified | This is the response code to an *If-Modified-Since* or *If-None-Match* header, where the URL has not been modified since the specified date. |
| 305 Use Proxy | The requested URL must be accessed through the proxy mentioned in the *Location* header. |
| 306 *Unused* | This code was used in a previous version. It is no longer used, but the code is reserved. |
| 307 Temporary Redirect | The requested page has moved temporarily to a new url. |

4xx: Client Error

|  |  |
| --- | --- |
| **Message** | **Description** |
| 400 Bad Request | The server did not understand the request. |
| 401 Unauthorized | The requested page needs a username and a password. |
| 402 Payment Required | *You can not use this code yet*. |
| 403 Forbidden | Access is forbidden to the requested page. |
| 404 Not Found | The server can not find the requested page. |
| 405 Method Not Allowed | The method specified in the request is not allowed. |
| 406 Not Acceptable | The server can only generate a response that is not accepted by the client. |
| 407 Proxy Authentication Required | You must authenticate with a proxy server before this request can be served. |
| 408 Request Timeout | The request took longer than the server was prepared to wait. |
| 409 Conflict | The request could not be completed because of a conflict. |
| 410 Gone | The requested page is no longer available . |
| 411 Length Required | The "Content-Length" is not defined. The server will not accept the request without it . |
| 412 Precondition Failed | The pre condition given in the request evaluated to false by the server. |
| 413 Request Entity Too Large | The server will not accept the request, because the request entity is too large. |
| 414 Request-url Too Long | The server will not accept the request, because the url is too long. Occurs when you convert a "post" request to a "get" request with a long query information . |
| 415 Unsupported Media Type | The server will not accept the request, because the mediatype is not supported . |
| 416 Requested Range Not Satisfiable | The requested byte range is not available and is out of bounds. |
| 417 Expectation Failed | The expectation given in an Expect request-header field could not be met by this server. |

5xx: Server Error

|  |  |
| --- | --- |
| **Message** | **Description** |
| 500 Internal Server Error | The request was not completed. The server met an unexpected condition. |
| 501 Not Implemented | The request was not completed. The server did not support the functionality required. |
| 502 Bad Gateway | The request was not completed. The server received an invalid response from the upstream server. |
| 503 Service Unavailable | The request was not completed. The server is temporarily overloading or down. |
| 504 Gateway Timeout | The gateway has timed out. |
| 505 HTTP Version Not Supported | The server does not support the "http protocol" version. |

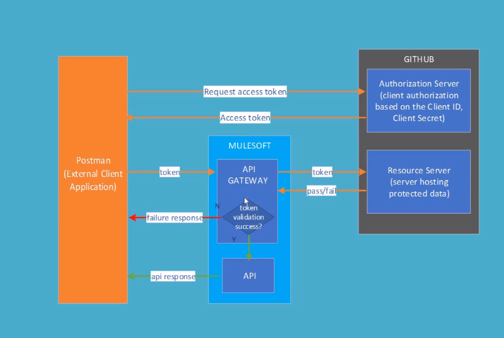
**28) OAUTH security:**

OAUTH2.0 is a security method that most of the API’s is using in order to protect he API from threats and to cater to only authentic requests.

A request coming from a client with a Bearer Token (OAUTH server says give the response to the bearer of the token) should be authenticated by the API Gateway Manager with the OAUTH server and once validated that the OAUTH server gave this valid token to this client by getting the client information of client id and client secret and if the token is valid then the Gateway API manager will open the request to the API and API will respond with the proper response to the client.

Now another concept of refresh token can also be considered where the client is given a a short lived token and a ling lived refresh token. Once the short lived access token expires the client can send the refresh token to the API gateway which in turn gets the access token from the OAUTH server and gets validated.

In Mule, we can enable an API with OAUTH 2.0 policy by giving the url of the OAUTH server and enable it. Once enabled the request from client has to come with an access token else the request will be denied.



Above picture says how OAUTH servers validates the token.