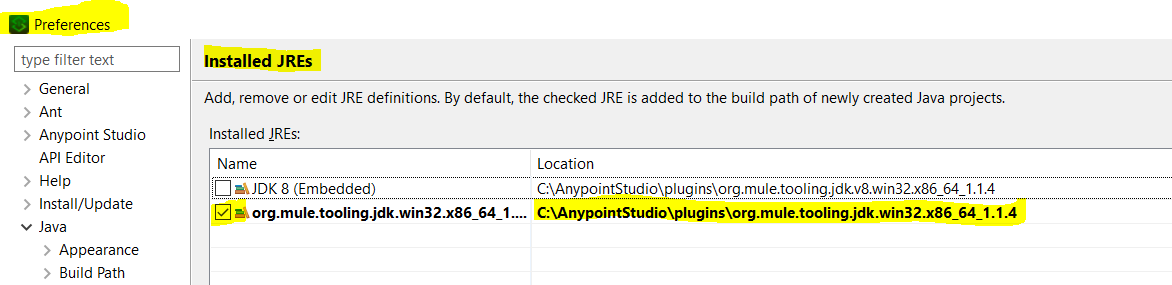
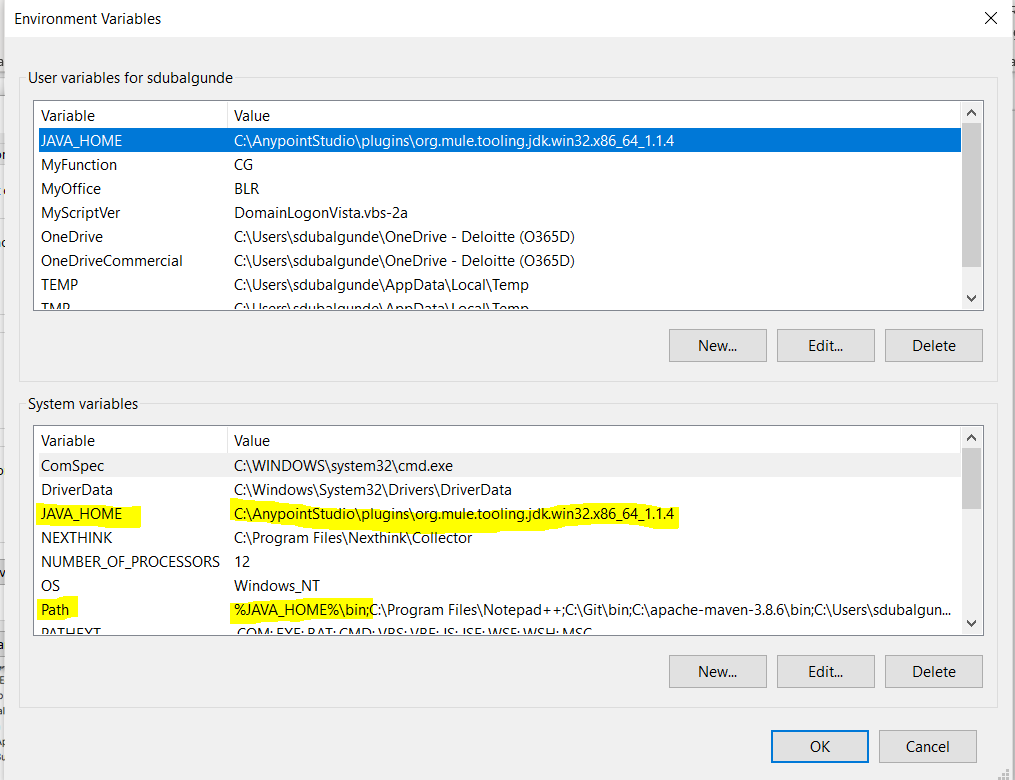
## **Prerequisites Tools**

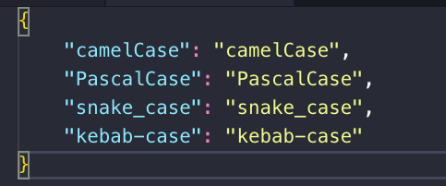
**REST Client** : [Postman](https://www.postman.com/downloads/) or Mulesoft’s Advanced Rest Client [ARC](https://install.advancedrestclient.com/install) used for making request

**Anypoint Platform Account:** For deploying ,managing API [Sign Up Here!](https://anypoint.mulesoft.com/login/#/signup?apintent=generic)

**Anypoint Studio:** Mulesoft’s IDE(Integrated Development Environment)to start building APIs and integrations quickly with hundreds of pre-built connectors, templates, and examples .[Download Here!](https://www.mulesoft.com/lp/dl/studio) N then setup path environment variables for %JAVA\_HOME%/bin n make sure that u r pointing the same jdk path for setting up path environment variables and anypoint studio’s windows->preferences->Installed JRE







* while defining project name,flow name,sub-flow name-private flow name,component t display name, .yaml properties ,mule .xml file names,global-config elements name use kebab-case like project name= hello-mule,flow name=hello-mule-main-flow,component name=sets-inputparams, mule .xml file=get-service-status-implementation.xml, global-config elements name=hello-mule-http-listener-config
* while Designing Raml for queryparams,headers,response data types and examples use snake\_case like queryparams= zip\_code,headers=source\_app, response data types=request\_tracking\_id,

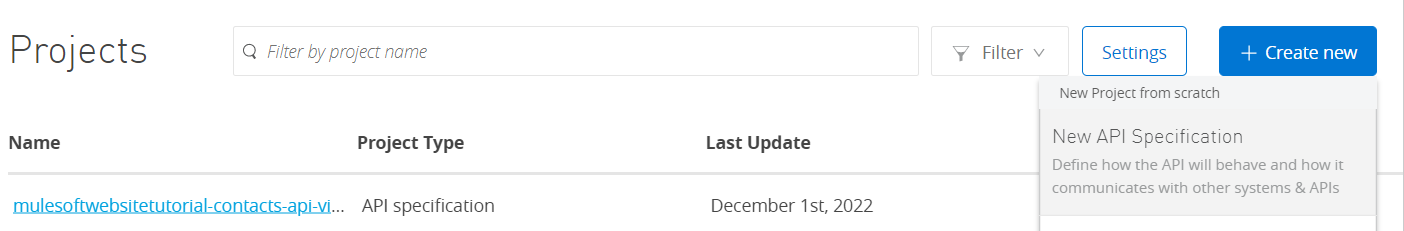
for uriparams,resources,Raml file-names use kebab-case like uriparams={customer-id}, resources=/service-status

* while defining variables inside anypoint studio use camelCase like serviceId,customerId

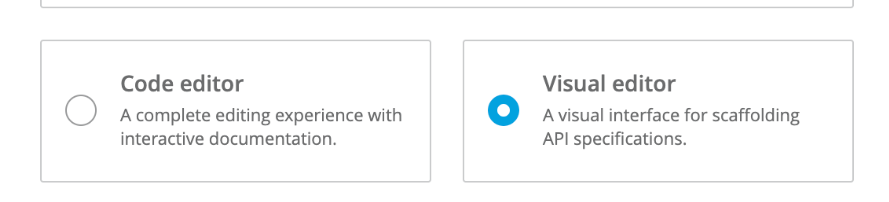
QUICKSTART

# Designing your first API specification

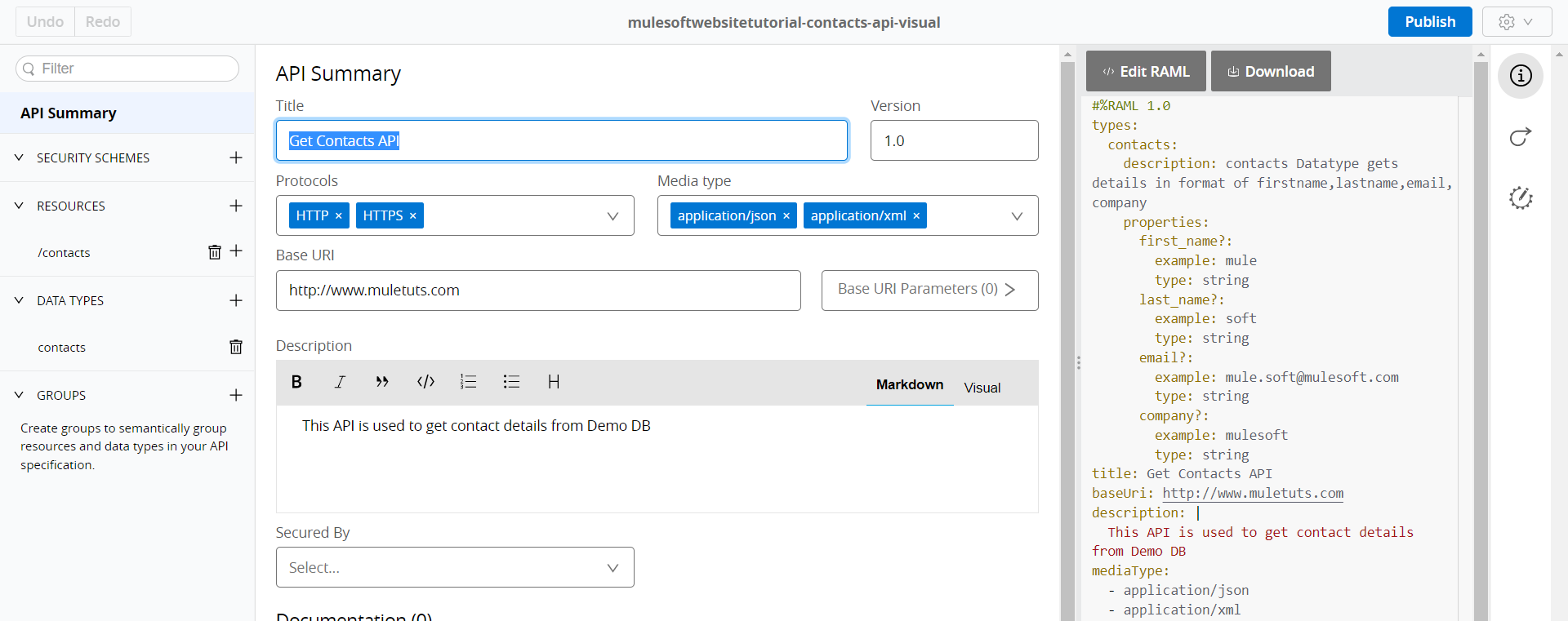
* API is nothing but a middleware which accepts the request n returns the response from the system. for eg: a waiter can be an API which accepts order from us n then takes that order to chef ,once shef prepares the order thee waiter returns us the food that we ordered.
* API Specification is a contract between consumer(who consumes the API) n provider(who provides the data within to API) which tells how API behaves n operates.
* The first step in the API development process is to brainstorm how we want our API to operate. Do we want our API to return a 200 valid response on POST requests and return a 405 method not allowed on GET requests? Do we want to design a RESTful API or a SOAP API? Do we want our API to be HTTP, HTTPS, or both? What data types do we want to accept in our API request? All of these questions are parameters that we can set while building our API within MuleSoft’s API Designer.
* For our first API, let’s create an integration that fetches a list of contacts from our internal database, and uploads them as leads into Salesforce. Once you are logged in to [Anypoint Platform](https://anypoint.mulesoft.com/login/" \t "_blank), navigate to **Design Center**. Click the **Create New** button, and select **New API Specification** as shown below.
* Scaffolding in software development generally means **creating a temporary structure for your project or setting everything up for your project**.



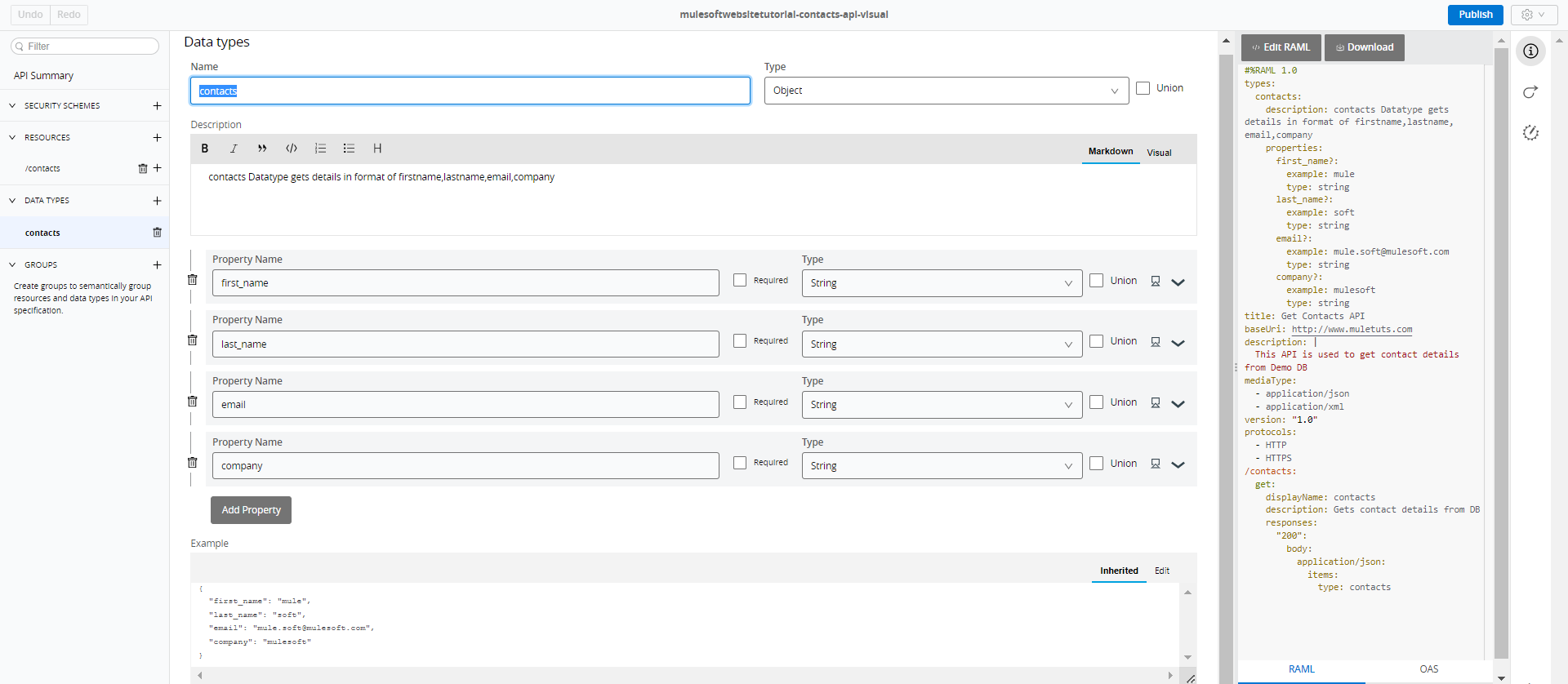
Creating API Spec in Visual Editor



Providing title,version,protocol,media-type,base uri,description,

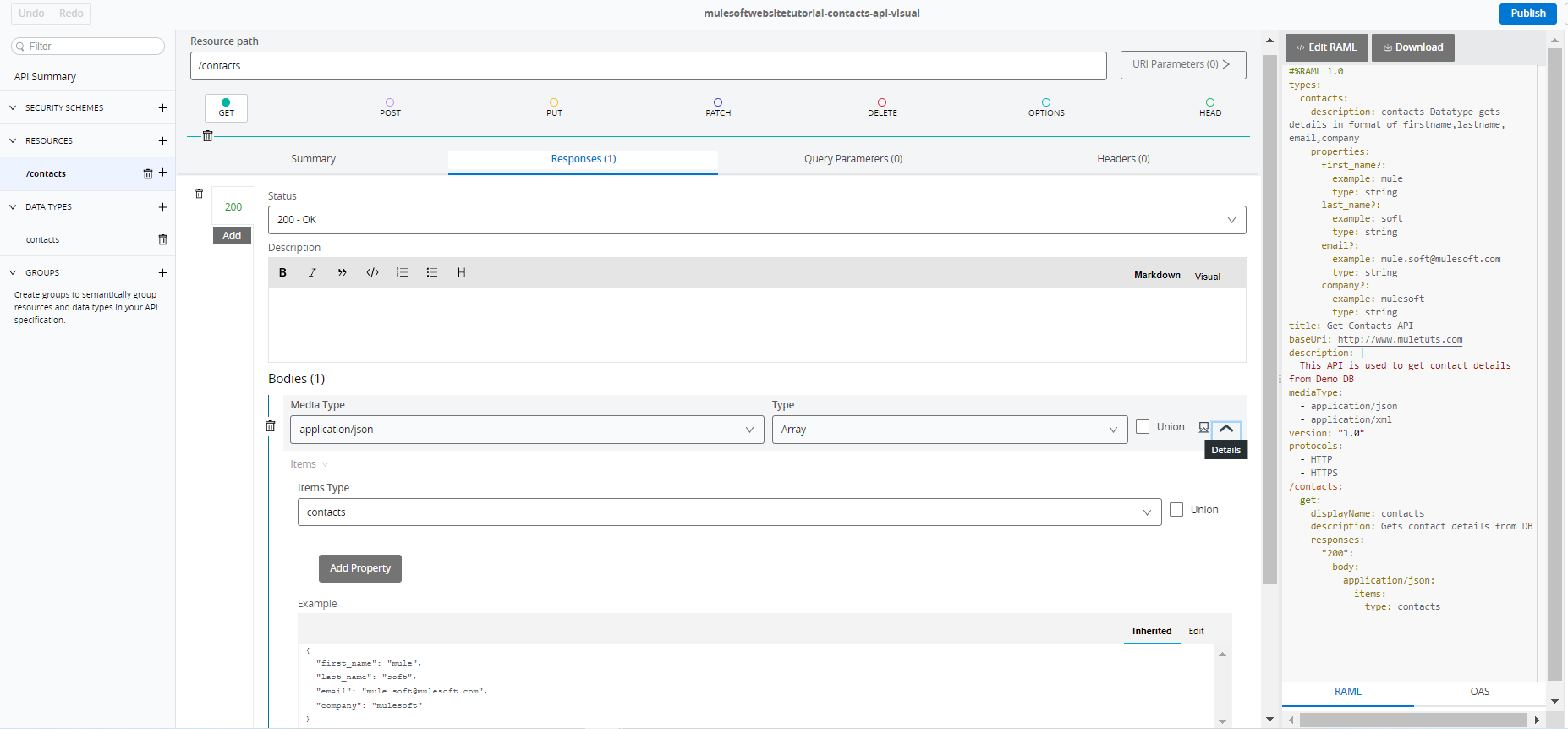


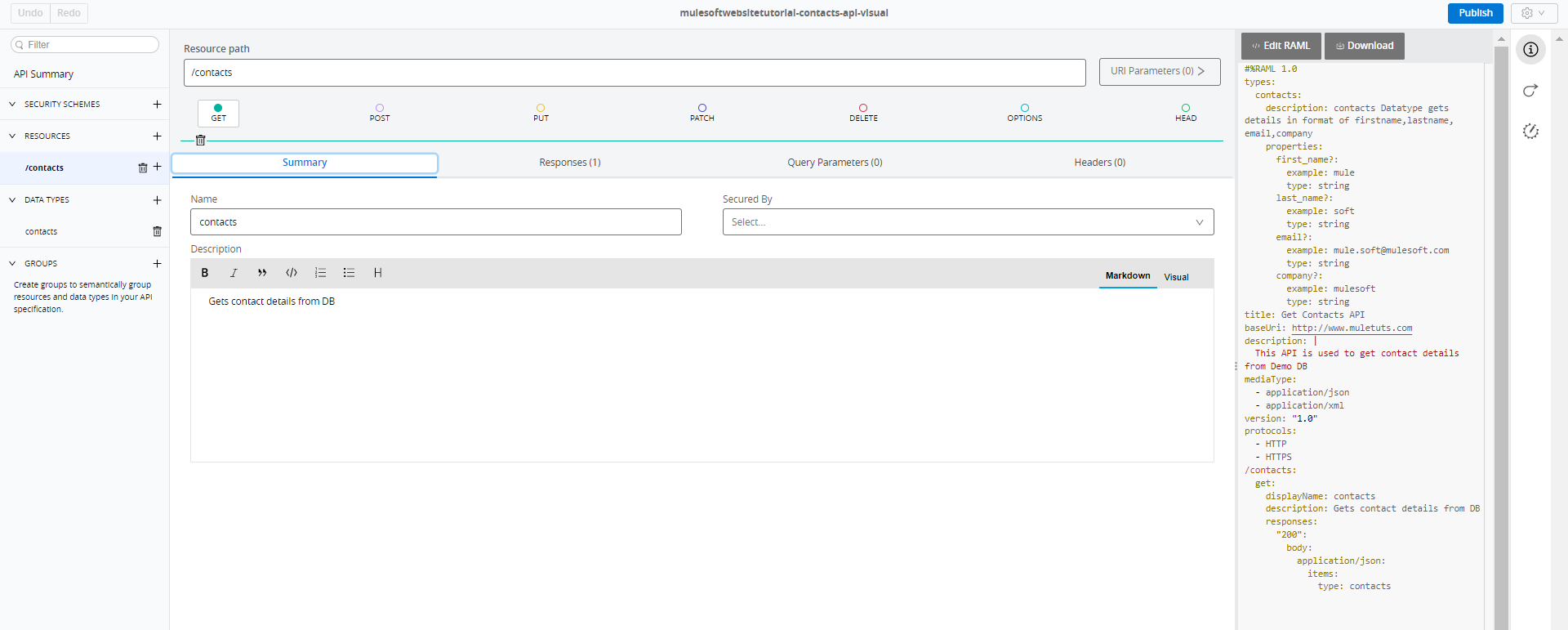
Creating datatype Contacts



* Creating a data type is helpful when creating flows since each card in the flow will automatically detect the formats and structures of the data that are in the inputs and outputs.

Creating resource(API Endpoint)

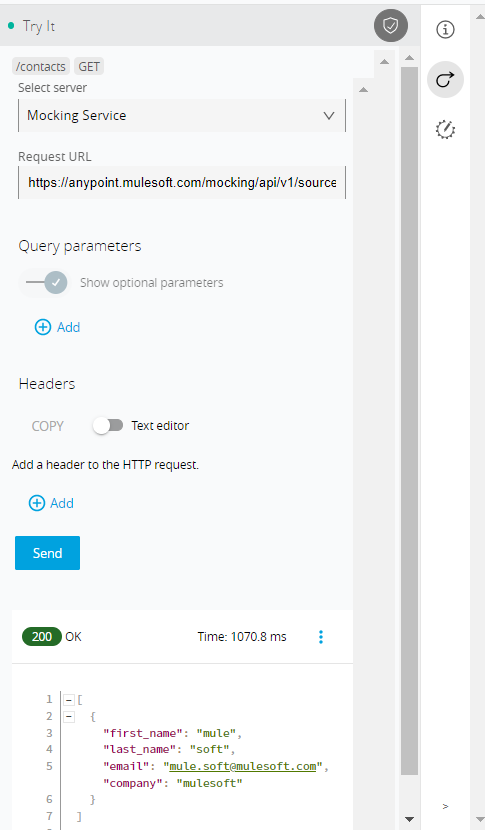


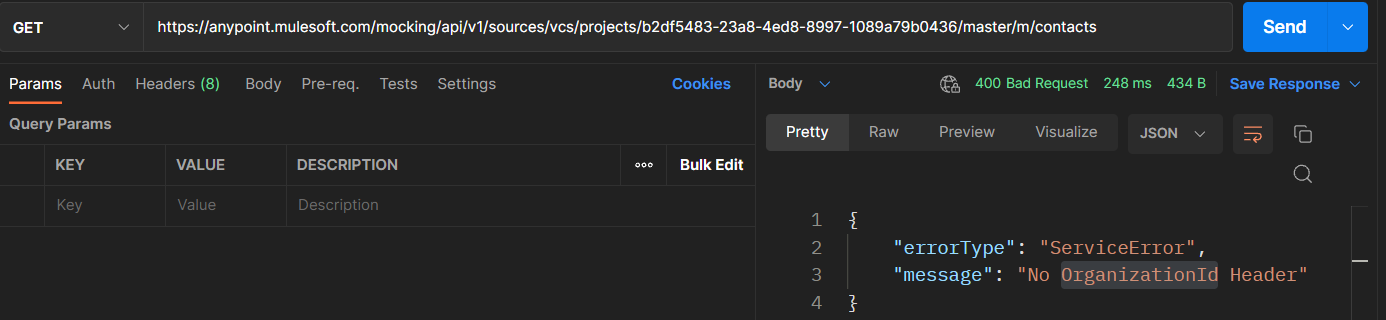


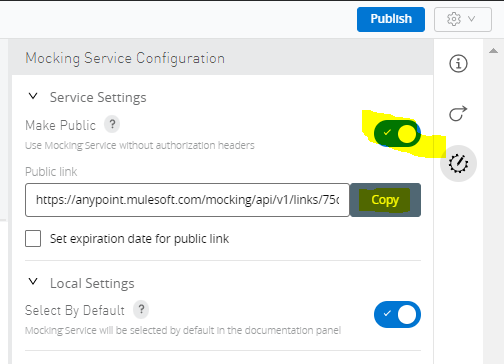
## **Test your API with the Mocking Service**

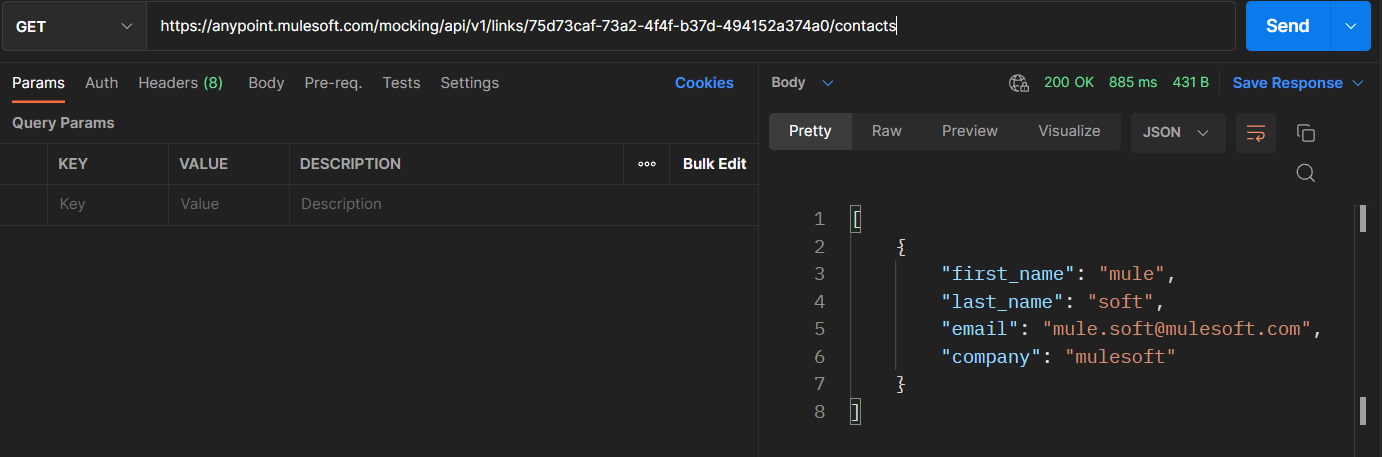
If we are testing RAML in Anypoint Platform’s Design Center with Mocking service enabled after we click on send we get the output but if we are testing the RAML with Postman or ARC then

We get 400 Bad Request with “No OrganizationId Header” becoz the URL that we are hitting will work only inside Anypoint platform n not outside the anypoint platform.to resolve this we have to use mocking service as public n copy the generated URL n then hit the endpoint





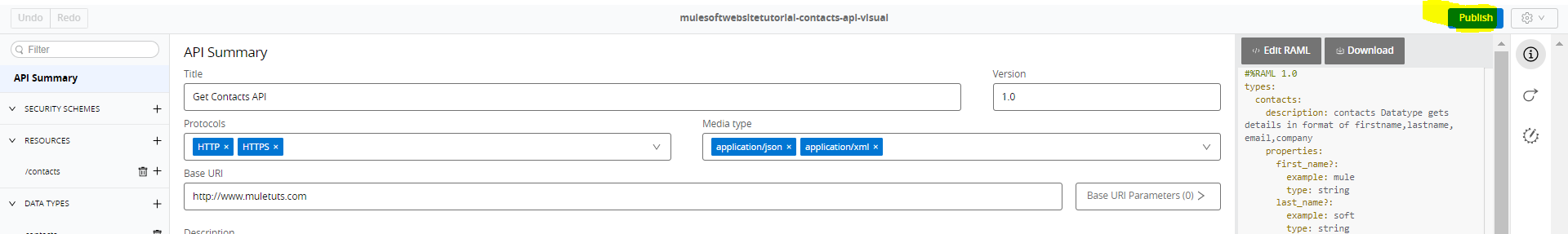




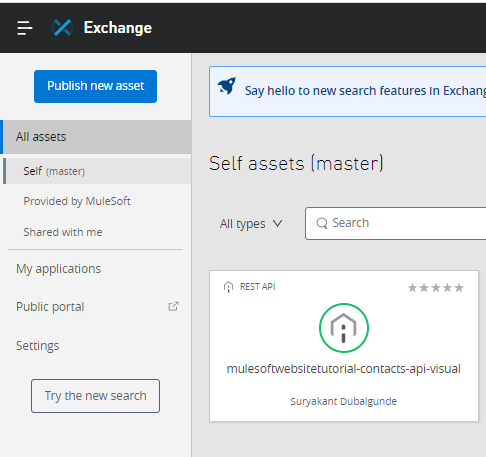
**After Raml Testing,Publish API into Exchange**

After Testing the RAML, we need to publish the API into exchange

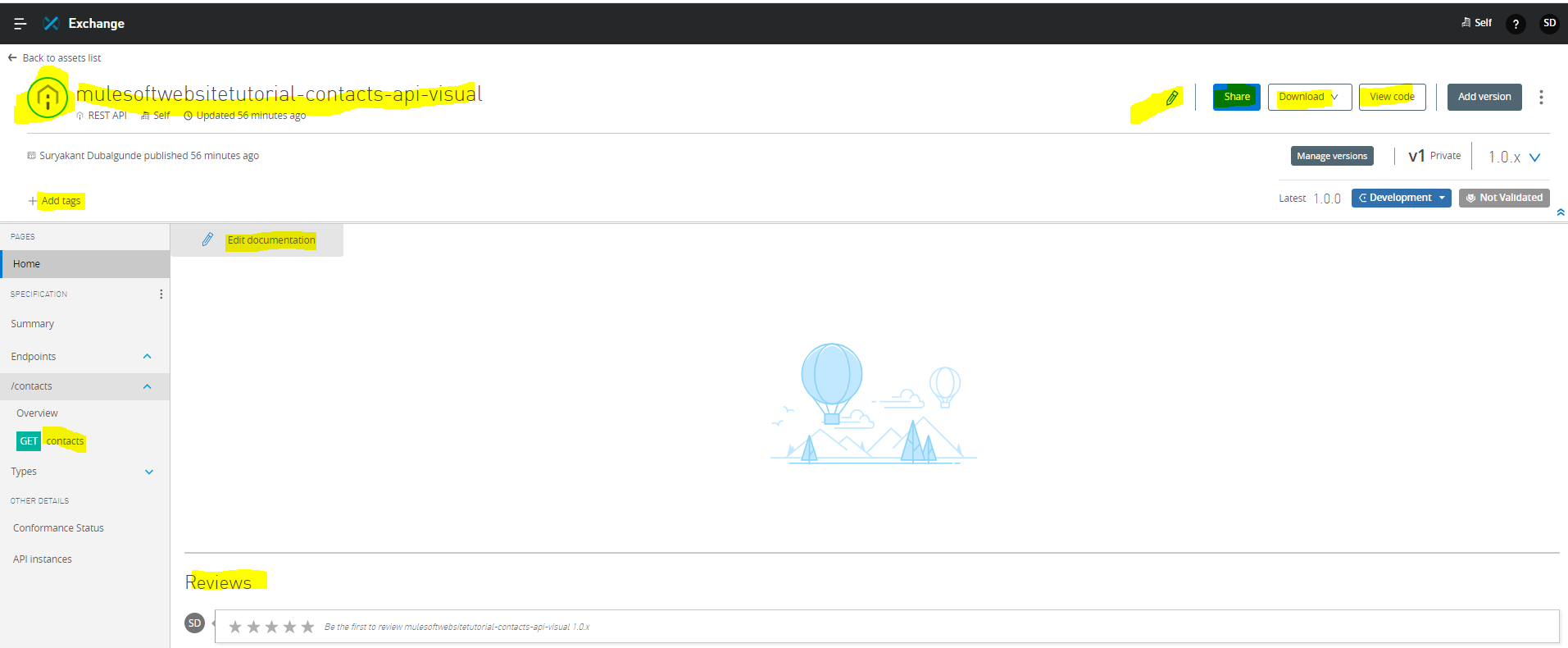
Exchange is a marketplace for connectors, APIs and templates that can be reused and shared across your organization. When you publish your API to Exchange, you can define whether you want your API to be private, public to anyone within your business organization, or public. Additionally, your API can be downloaded and reused in any of your future integrations or projects. To publish your API, simply click **Publish** at the top right of the screen in Design Center and select **Publish to Exchange**.



After publishing, you will see your API listed in the Exchange which means you can now import your API into your Anypoint Studio project to build your first integration.



In Exchange, you can add a photo, edit the title, description and even rate and comment on your published API.



Visual Editor Code Code Editor Code(simple)

Notes:

* RAML stands for RESTful API Modeling Language , is a YAML-based language for describing RESTful APIs.
* YAML stands for Yet Another Markup Language or YAML Ain’t Markup Language
* In RAML’s root file,first line is

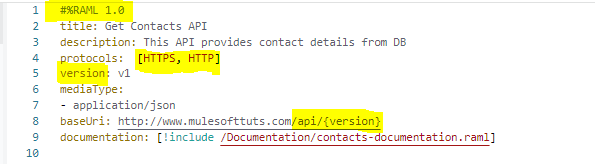
#%RAML 1.0

Which is commented(#=symbol) n indicates RAML version we are using.

* In RAML,we can have single line comment only by using # symbol.multi-line comment is not supported in RAML as It uses YAML.

# This is comment line

* common-declarations In RAML are , Raml version indicator,title,description,version,protocols,mime-type,baseuri,documentation etc.



Notice on line 8 the use of braces { } around the word “version“. This is how we tell RAML that “version” refers to a property and is to be expanded. Therefore the actual baseUri will be: [**http://www.mulesofttuts.com/api/v1**](http://www.mulesofttuts.com/api/v1)

[Note: the *version* property is optional and need not be a part of the *baseUri*.]

* In YAML, we can show the list of params either with –(dash) or with including the list in an [] (square brackets with comma-separated.

Like below:

Protocols:

* HTTPS
* HTTP

Or

Protocols: [HTTPS, HTTP]

documentation:

  - !include /Documentation/contacts-documentation.raml

OR

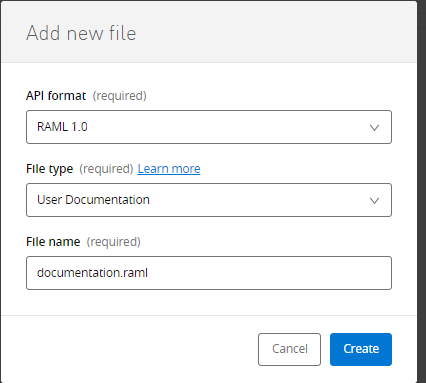
documentation: [!include /Documentation/contacts-documentation.raml]

* For involving externalize declared file in RAML’s root file or any other file, we can use

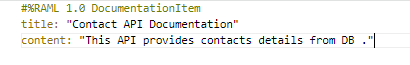
!include <path-of-file>

Like above documentation example

To create Documentation externalize file we need to click + sign and new file ->select file type as “User Documentation”->give filename n select create



Will create a file n shows commented RAML vesion with Filetype as “DocumentationItem” with title n content



To refer this externalize documenationItem file in main root file or any other file we will use we can use !include <path-of-file>

documentation:

  - !include /Documentation/contacts-documentation.raml

OR

documentation: [!include /Documentation/contacts-documentation.raml]

To create diff.types of file

|  |  |  |
| --- | --- | --- |
| File Type | Purpose of File | File Content |
| User Documentation | To provide brief description about API |  |
| Data Type | To provide datatypes of request/response of API |  |
| Example |  |  |
| Trait |  |  |
| Resource Type |  |  |
| Library |  |  |
| Security Scheme |  |  |

* A resource /resource is called as API Endpoint.
* Datatype n examples are declared to predict format of the input/output in anypoint studio

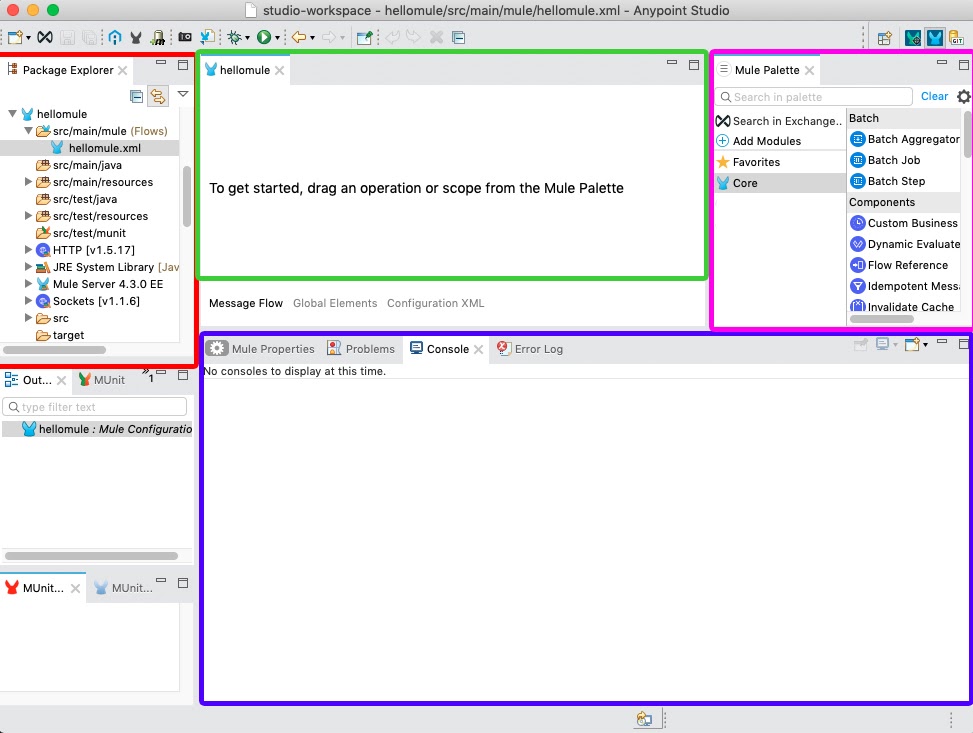
GETTING STARTED

1.Build your first Hello Mule application

Once launched Anypoint Studio, click on **File** then select **New > Mule Project**.

Anypoint Studio will open the **New Mule Project** wizard. In the **Project Name** field, enter the value HelloMule, then click **Finish**.

Anypoint Studio will open a new blank project. The following example below explains the different views you will use to build a Mule project:



| **Color** | **Area Name** | **Description** |
| --- | --- | --- |
| Red | Package Explorer | This is where you can view your project structure (files and folders). |
| Blue/Purple | Properties Editor and Console | This is where you can configure attributes of connectors and see the logs to catch any errors in your application. |
| Pink | Mule Palette | The Mule Palette allows you to select from hundreds of prebuilt Anypoint Connectors from Exchange or the Core Connector components to build your application. |
| Green | Canvas | This is where you can drag and drop Modules located in the Mule Palette to create a message flow. |

Http Listener:-An HTTP Listener is an HTTP endpoint that listens for an HTTP request to come to the URL you define. When the Listener receives an HTTP request, the contents inside of the flow will execute in the order that you define.

If we specify a basepath in http-listener,then while hitting the API Request we must use that basepath otherwise will get an error like “no listener for endpoint /resource-name” with 404 Not Found http status code

To refer any config.yaml property in anypoint studio use ${property-name} .incase of mule Expression button fx is available then try p(‘property-name’)

uriParams and queryParams:

uriParams is required part of url n queryparams is optional part

uriParams is a value n queryparams is key=value pair

uriparams is separated by / from url n queryparams is separated by ?

multiple uriparams can merged and specified by using /

multiple queryparameters can merged and specified by using &

if uriparams is not specified in the request then we can’t hit the listener n listener will fail with “no listener for endpoint error” with 404 not found

Deployment step field:In runtime manager while deploying the App we get below fields

By default sandbox environment is available for deployment in trial account but design is not

1. Application Name: it is required n should contain only lowercase letters,numbers & hyphen(-) n should be minimum 3 & max 42 characters n should not start or end with hyphen(-) n it should be available for use
2. Deployment target: based on target available to deploy it shows the values in Runtime manager.like cloudhub 2.0,cloudhub,hybrid etc.
3. Application File: the .jar or .zip file
4. Runtime: specifies runtime version to use for deployment with no of workers ,worker size

Worker Size: Memory Name

* 1. 500Mb micro
  2. 1Gb small

1 1.5Gb medium

2 3.5Gb large

4 7.5Gb xlarge

8 15Gb xxlarge

16 32Gb 4xlarge

|  |
| --- |
| No of workers |
| 1 |
| 2 |
| 4 |
| 8 |
| 16 |

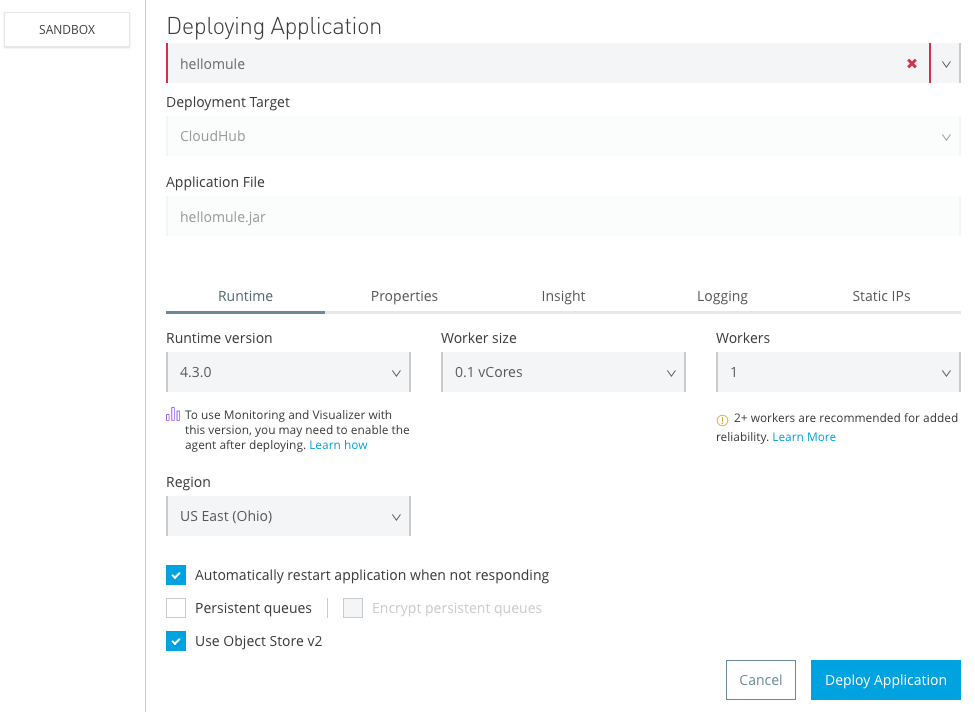
1. Properties: here we can specify the properties which are needed to run the app n these are runtime properties so we can change the property value at any time.
2. Insight: stores metadata by default it is disabled
3. Logging:
4. Static Ip: we can use this option if mule app is not able to connect with on-prem system so we can share the static ip generated to the system team to provide the access to the ip so that mule app deployed on cloud can access on-prem system.

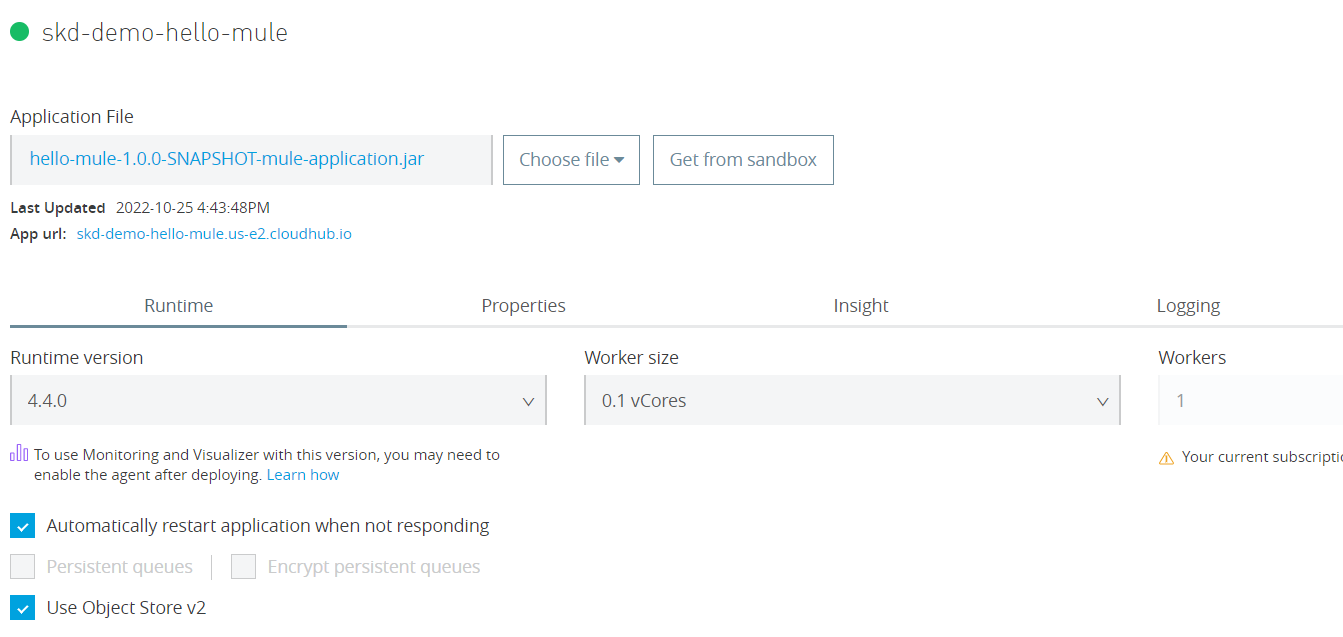
CloudHub will issue your application with a publicly accessible endpoint URL after you complete the deployment process. Step:-

Right-click on your project in your Package Explorer, and click on **Anypoint Platform > Deploy to CloudHub**.

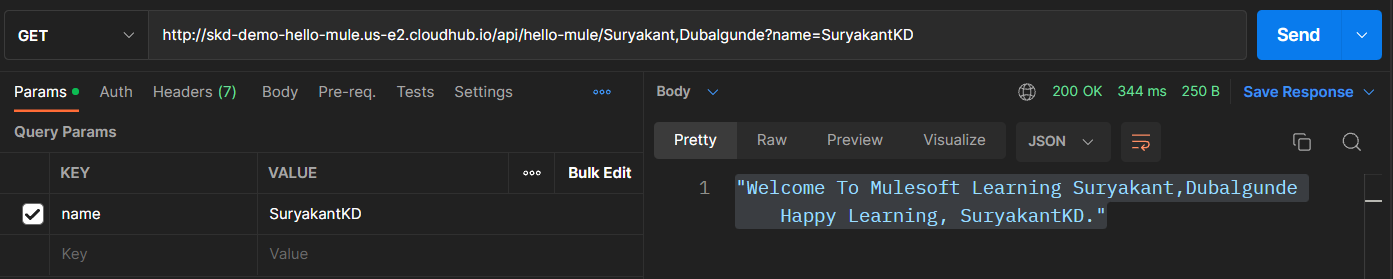
Or

Export the jar file n go to **Anypoint Platform >Runtime Manager>Deploy Application.**





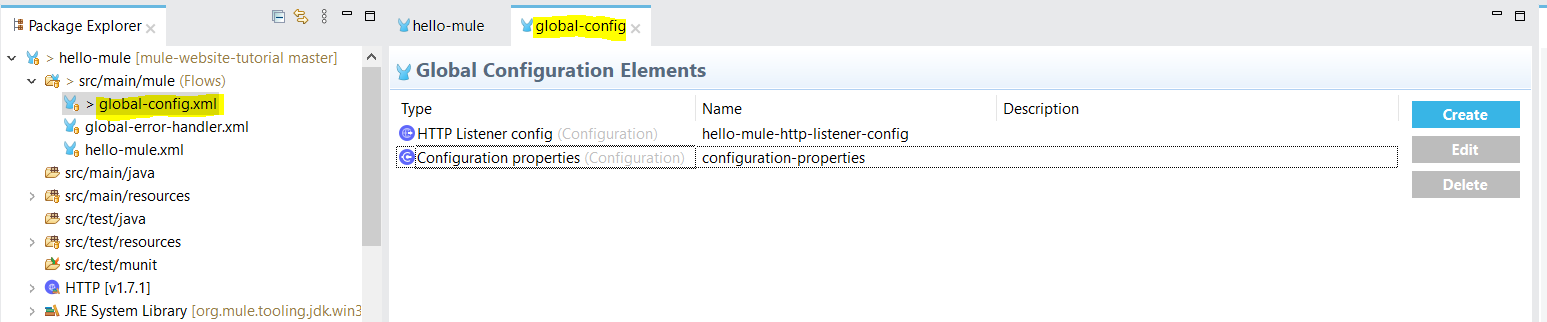
To access this mule app thru postman copy the App Url n paste it in place of <http://host:port>



# How to set up your global elements and properties files in Anypoint Studio

## **1.Create a global-config.xml file to keep the global elements**

Right-click on the **src/main/mule** folder. Select **New > Mule Configuration File**. Give name as global-config.xml and use this file for storing all the global element configurations



## **2.Externalize the hardcoded values into properties**

To avoid having hardcoded values in our code, it is a best practice to externalize them into properties files. Right-click on the **src/main/resources** folder and select **New > File**.

There are two types of supported properties files: they should be defined under src/main/resources folder.

* .properties
* .yaml
* File Properties:- **${file::properties-file.txt}** The placeholder value can also be the entire content of the file. The placeholder value becomes the string value,

for example: sql-query.txt

SELECT

e.employee\_id AS "Employee #"

, e.first\_name || ' ' || e.last\_name AS "Name"

, e.email AS "Email"

, e.phone\_number AS "Phone"

, TO\_CHAR(e.hire\_date, 'MM/DD/YYYY') AS "Hire Date"

, TO\_CHAR(e.salary, 'L99G999D99', 'NLS\_NUMERIC\_CHARACTERS = ''.,'' NLS\_CURRENCY = ''$''') AS "Salary"

, e.commission\_pct AS "Comission %"

, 'works as ' || j.job\_title || ' in ' || d.department\_name || ' department (manager: '

|| dm.first\_name || ' ' || dm.last\_name || ') and immediate supervisor: ' || m.first\_name || ' ' || m.last\_name AS "Current Job"

, TO\_CHAR(j.min\_salary, 'L99G999D99', 'NLS\_NUMERIC\_CHARACTERS = ''.,'' NLS\_CURRENCY = ''$''') || ' - ' ||

TO\_CHAR(j.max\_salary, 'L99G999D99', 'NLS\_NUMERIC\_CHARACTERS = ''.,'' NLS\_CURRENCY = ''$''') AS "Current Salary"

, l.street\_address || ', ' || l.postal\_code || ', ' || l.city || ', ' || l.state\_province || ', '

|| c.country\_name || ' (' || r.region\_name || ')' AS "Location"

, jh.job\_id AS "History Job ID"

, 'worked from ' || TO\_CHAR(jh.start\_date, 'MM/DD/YYYY') || ' to ' || TO\_CHAR(jh.end\_date, 'MM/DD/YYYY') ||

' as ' || jj.job\_title || ' in ' || dd.department\_name || ' department' AS "History Job Title"

FROM employees e

-- to get title of current job\_id

JOIN jobs j

ON e.job\_id = j.job\_id

-- to get name of current manager\_id

LEFT JOIN employees m

ON e.manager\_id = m.employee\_id

-- to get name of current department\_id

LEFT JOIN departments d

ON d.department\_id = e.department\_id

-- to get name of manager of current department

-- (not equal to current manager and can be equal to the employee itself)

LEFT JOIN job\_history jh

ON e.employee\_id = jh.employee\_id

-- to get title of job history job\_id

LEFT JOIN jobs jj

ON jj.job\_id = jh.job\_id

-- to get namee of department from job history

LEFT JOIN departments dd

ON dd.department\_id = jh.department\_id

ORDER BY e.employee\_id;

Get-service-status-implementation.xml

**<mule:set-payload value="${file::** **sql-query.txt}"/>**

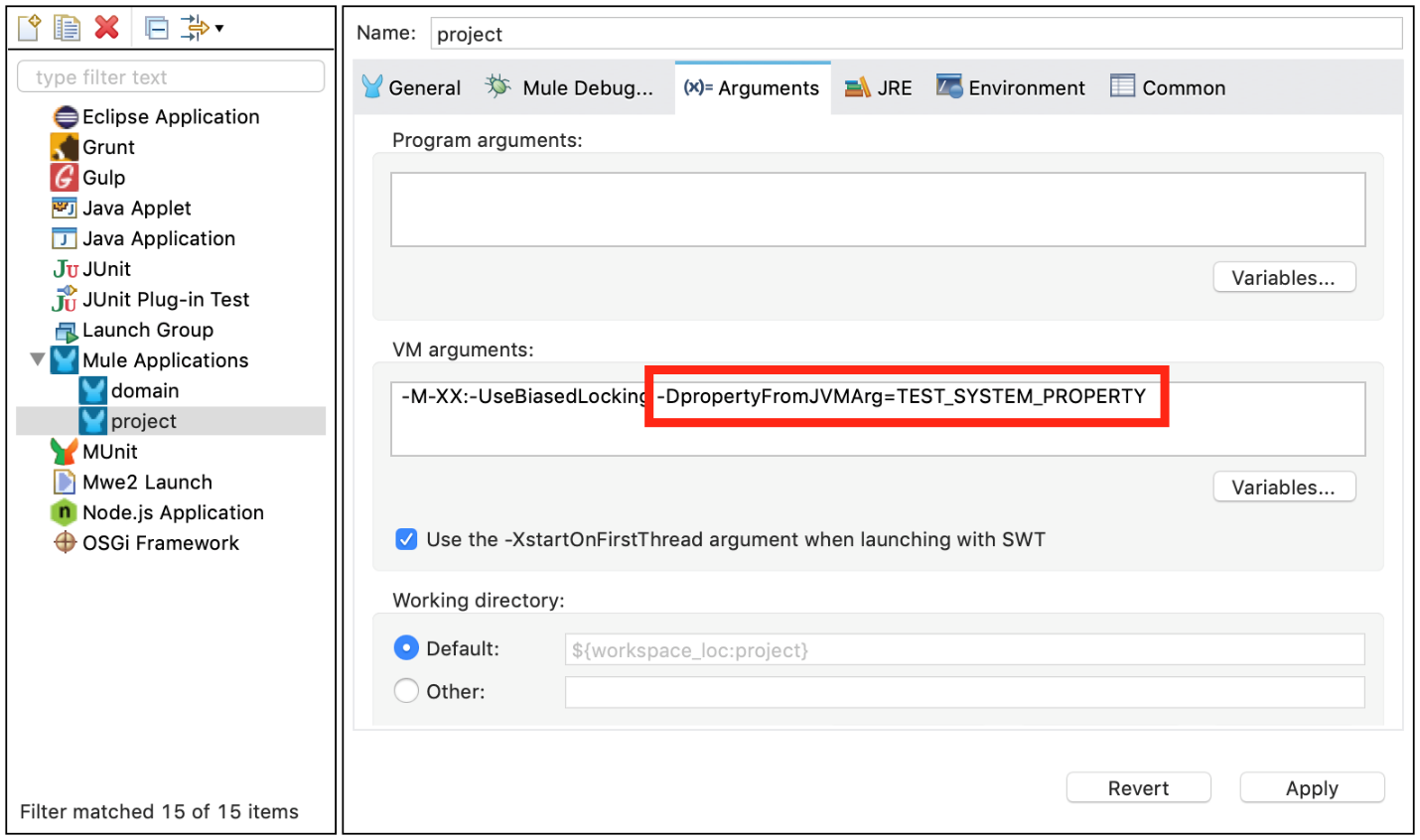
The payload’s value becomes  “select .............. ORDER BY e.employee\_id:” Just like other properties files, these files must be located in src/main/resources, inside your Mule project.

This practice is useful for modularizing the configuration file: You can extract large contents from the config file, SQL queries, or transformations to make the config file clearer, and you can reuse the contents.

### Setting System Properties in Anypoint Studio

You can add properties when you launch your project on Anypoint Studio, through the Run Configurations menu:

1. Right-click your project in Package Explorer.
2. Click **Run As** > **Run Configurations**.
3. Pick the **Arguments** tab.
4. Add your arguments to the **VM arguments** field, preceding property names with **-D**:



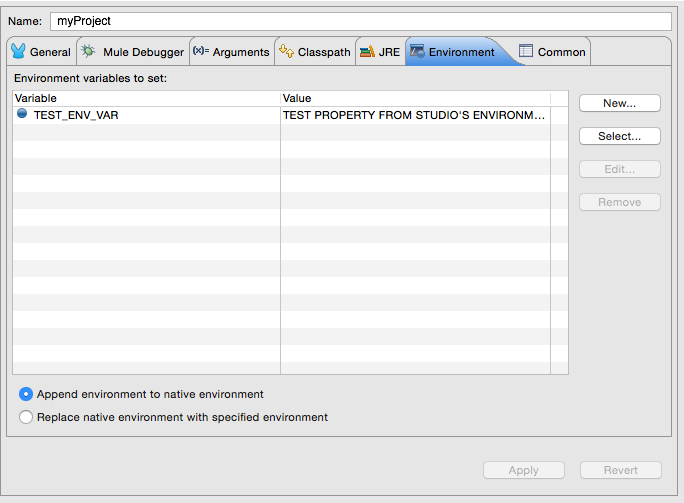
Your properties are now available each time you deploy your app through Studio. You can then reference them with the following syntax:

**<logger message="${propertyFromJVMArg}" doc:name="System Property Set in Studio through JVM args"/>**

### Setting Environment Variables in Anypoint Studio

You can set variables in Studio through the Run Configuration menu:

1. Right-click your project in Package Explorer.
2. Select **Run As** > **Run Configurations**.
3. Pick the Environment tab.
4. Click the **New** button and assign your variable a name and value.



Your variable is now available each time you deploy through Studio. You can reference it with the following syntax:

**<logger message="${TEST\_ENV\_VAR}" doc:name="Environment Property Set in Studio"/>**

## Setting the Properties File Dynamically

A common configuration use case is to set the file to depend on a property (for example, env) to determine which file to use, for example, to use a development-properties file in development stage or a production file.

**<configuration-properties file="${env}-properties.yaml"/>**

XML✓ Copied

This way, the value of the property env determines which file to use to load the configuration properties. That env property can be set by a global property, system property, or environment property. You can use global properties as a way to define default values for configuration properties. System and environment properties with the same name as a global property will override that global property.

**<global-property name="env" value="dev"/>**

**<configuration-properties file="${env}-properties.yaml"/>**

This way, the default value for the env property is dev, which can still be overridden with a system or environment property. Please note that this configuration is **required** for metadata resolution in Anypoint Studio. If you do not define default values for the properties that are passed through the command line, you receive an error while creating an application model for all message processors that depend on them.

Another thing to consider is that placeholders of a configuration property setting cannot depend on the properties loaded from another configuration property. In the example above, the property env couldn’t have been defined in a configuration property. The example below **is not correct**:

**<configuration-properties file="file-with-env-property.yaml"/>**

**<configuration-properties file="${env}-properties.yaml"/>**

## Setting Properties Values in Runtime Manager

If you deploy your application to [Runtime Manager](https://docs.mulesoft.com/runtime-manager/), you can also set properties through the Runtime Manager console. These can be defined when [Deploying to CloudHub](https://docs.mulesoft.com/runtime-manager/deploying-to-cloudhub), or on an [already running application](https://docs.mulesoft.com/runtime-manager/managing-applications-on-cloudhub).

To create an environment variable or application property:

1. Log in to your Anypoint Platform account.
2. Click Runtime Manager.
3. Either click **Deploy Application** to deploy a new application, or select a running application and click **Manage Application**.
4. Select the **Properties** tab in the **Settings** section.

## Properties Hierarchy

Configuration properties can be overwritten. The Mule runtime engine uses the following hierarchy to determine which properties take precedence if they have the same name. In this hierarchy, deployment properties are at the top, so they take precedence over the other types of properties.

1. Deployment properties
2. System properties
3. Environment properties
4. Application properties (includes configuration properties, secure configuration properties, and other custom configuration properties)
5. Global Properties

So, for example, if a configuration property named size is defined as a system property, and there is also an application configuration property named size, the value for the application is the value of the property with the most precedence (in this case, the system property).

Also, a property can derive its value from other properties that are higher in the hierarchy. Therefore, an application property can derive its value from environment, system, and deployment properties. A system property can derive its value from a deployment property, and so on. For example, if there is a system property named env, an application configuration property can have the value file.${env}.xml. However, an application property cannot depend on an application property’s value unless it’s defined in the same file. For example, a Secure Configuration property cannot depend on a Configuration property.

Because of this hierarchical structure, Mule components that load files cannot depend on an application or a global property. Mule runtime engine loads both, the component and the application property, at the same hierarchy. For example, if Mule loads an import component before loading the application property, the deployment fails because the property is not resolved by the time the import component needs to load.

## Built-in Properties

Mule runtime engine defines internal Application properties and assigns a value to them based on the configuration of the Mule instance and the deployed applications:

| **Property** | **Description** |
| --- | --- |
| app.home | The path to the Mule application /home directory |
| app.name | The name of the Mule application |
| domain.name | The name of the Domain |
| api.name | The name of the API |
| mule.home | The path to the Mule instance /home directory |

Do not overwrite any of these properties; Mule uses some of these values internally and overwriting them might cause unexpected behaviors.

\ is the escape character used in mule

You can also escape the resolution of property placeholder values:

<logger message="\${property.value}"/>

Instead of returning the value of the property, the Logger component returns the string "${property.value}".

A mule app can have multiple-declarations of configuration-properties as a global-element in global-config.xml indicating diff. properties file. Like below:

<configuration-properties file="http.yaml"/>

<configuration-properties file="smtp.yaml"/>

<configuration-properties file="db.yaml"/>

Let say if we have one property called port in all 3 above declared files

http.yaml

port: “8091”

smtp.yaml

port: “8081”

db.yaml

port: “1521”

then for http -listener,smtp connector or db-connector we are using ${port} so in this situation mule will consider http.yaml’s port: “8091” becoz it is declared 1st in order so .

## **Create your environments’ secure properties files**

To store the global variables, unique tokens or keys, or login credentials,private keys we can store them in properties file.

it is a best practice to separate your secure properties in files per environment. You can create a **local.secure.properties** file for your local settings, a **dev.secure.properties** for your dev environment, a **qa.secure.properties** for your testing environment, etc.

To create a local.secure.properties file, right-click on **src/main/resources** and click on **New > File** and name the file local.secure.properties.

|  |  |
| --- | --- |
| 1  2 | **db.username**=myUsernameLocal  **db.password**=myPasswordLocal |

The next step is to search Exchange for the **Mule Secure Configuration Properties** module. In your **global.xml** file, select the **Search in Exchange** option from the Mule Palette. Search for the module and click on **Add >**.