MuleSoft Certified Developer - Level 2 (Mule 4)

Expose production-ready Anypoint Platform-managed APIs from Mule applications

- Implement versioning of specific API-related artifacts
- Configure custom or out-of-the-box (OOTB) API policies
- Implement server-side caching of API invocations using API policies
- Request access to APIs while honoring environment-specific scoping
- Implement HTTP callbacks (webhooks)
- Code API implementations to perform API auto-discovery

Implement performant and reliable Mule applications

- Implement ObjectStore persistence for all Mule deployment options
- Implement fault-tolerant, performant, and traceable message passing with the VM and AnypointMQ connectors
- Implement fault-tolerant invocations of HTTP-based APIs, reacting correctly to HTTP status codes
- Validate assertions using the Validation module
- Validate messages against XML- or JSON-Schema documents
- Parallelize integration logic using scatter-gather
- Implement compensating transactions for partially failed scatter-gather
- Implement client-side caching of API invocations for performance

Implement monitorable Mule applications

- Expose healthcheck endpoints from a Mule application
- Implement effective logging
- Change log levels and aggregate and analyze logs
- Monitor Mule applications and Mule runtimes using Anypoint Platform or external tools
- Implement message correlation, including persistence and propagation of correlation IDs over HTTP

Implement maintainable and modular Mule applications and their Maven builds

- Modularize and optimize Mule application Maven build configurations
- Implement Maven-based automated deployment to Mule runtimes
- Execute MUnit tests with Maven
- Implement unit tests with the MUnit framework and tooling
- Build custom API policies
- Encapsulate common Mule application functionality in libraries
- Implement custom message processors using the Mule SDK or XML SDK

Secure data at rest and in transit

- Implement secure, environment-dependent properties management
- Create, package, and distribute keys and certificates
- Expose and invoke APIs over HTTPS
- Implement TLS mutual authentication on the client and server side
- Implement API invocations authenticated by Basic Auth or OAuth2 with HTTP or REST connectors

MuleSoft Certified Developer – Level 1

MuleSoft Certified Platform Architect – Level 1

MuleSoft Certified Integration Architect – Level 1

MuleSoft Certified Developer - Level 2

2 hours Virtual

Summary

A *MuleSoft Certified Developer – Level 2* should be able to independently work on production-ready Mule applications – applications that are ready to be used in a DevOps environment in professional software development projects and address and balance critical non-functional requirements including monitoring, performance, maintainability, reliability, and security. The *MuleSoft Certified Developer – Level 2* exam validates that a developer has the required knowledge and skills to:

Expose production-ready Anypoint Platform-managed APIs from Mule applications

Implement maintainable and modular Mule applications and their Maven builds

Implement monitorable Mule applications

Implement performant and reliable Mule applications

Secure data at rest and in transit

Topics Covered in Level 2 training

Section 0	**Local System Environment Setup and Backend Services Setup** *Install Software | Anypoint Platform | Import & Publish RAML | Build Backend Mule Apps | CloudHub Deploy | API Portal*
Section 1	**Setting Project Structure, Deployment Strategy and Development Best Practises** *OAS | API AutoDiscovery | One-way TLS | CloudHub Architecture | Secure Properties | Maven Deploy | Dependency Management*
Section 2	**Mule Integration - API Development, Error Handling, Caching Strategies** *Non-functional Requirement | OAuth Invocation | Error Handling | Client & Server Side Caching Techniques | Parallel Execution*
Section 3	**Asynchronous Message Processing, Logging Options, Tracing, Correlation IDs and Content Validation** *HTTP Callbacks | VM Queue & DLQ | Anypoint MQ | ACK | NACK | Circuit Breaker | Logging | Tracing | CorrelationID | Validations*
Section 4	**Mule Application Health Checks and Custom Connector Development** *Liveness and Readiness Concepts | Mule App Health Endpoints | Shared Library | Monitoring | XML SDK | Exchange Deployment*
Section 5	**Custom API Policy Development, Offline Policies Management** *Generate Custom API Policy Skeleton | Develop Custom API Policy | Manage Offline Policies | Deploy to Exchange | HandleBars*
Section 6	**Consuming SOAP Web Services using TLS Certificates (one-way and two-way)** *Use Mule 4 Standalone | Consume SOAP Web Service using plain HTTP | Consume SOAP Web Service using 1-way-TLS and 2-way-TLS*
Section 7	**MUnit Testing** *Configure MUnit for Mule | Prepare Test Data | Mock | Assert | Spy | Verify | Errors | Refactor Logic | Automated MUnit Recorder*

Setting up Expectations

What is Covered/Provided?

All topics are covered with Hands-On exercises

- Anypoint MQ > **Slides, Videos and Code Snippets**
- Anypoint Functional Monitoring > **Slides**
- MUnit Maven Execution > **Slides and Pointers**

What is NOT Provided?

Mule Enterprise Maven Repo credentials

- Anypoint MQ > **No Hands-On**
- Anypoint Functional Monitoring > **No Hands-On**
- MUnit Maven Execution > **No Hands-On**

Work on them, if you have ACCESS to these resources

#0 Local System Environment Setup and Backend Services Setup

- Software Installation
- Understand the Course Usecase & Architecture
- Create Anypoint Platform account and set it up
- Import RAMLs and Publish them to Exchange
- Add API Instances in API Manager
- Add Automated Policy
- Import RAML to Studio and make changes
- Deploy Application to Cloudhub
- Expose the API to Public
- Access API using credentials from Client Application in API Portal

0.1 Software Installation

- OpenJDK
- Maven
- Anypoint Studio
- Mule 4 Standlaone
- SoapUI
- Postman
- Git Bash

Setup Local Environment

OpenJDK8

Mule 4
Standalone

maven
3.6.x or later

git
bash

7.11.1 or later

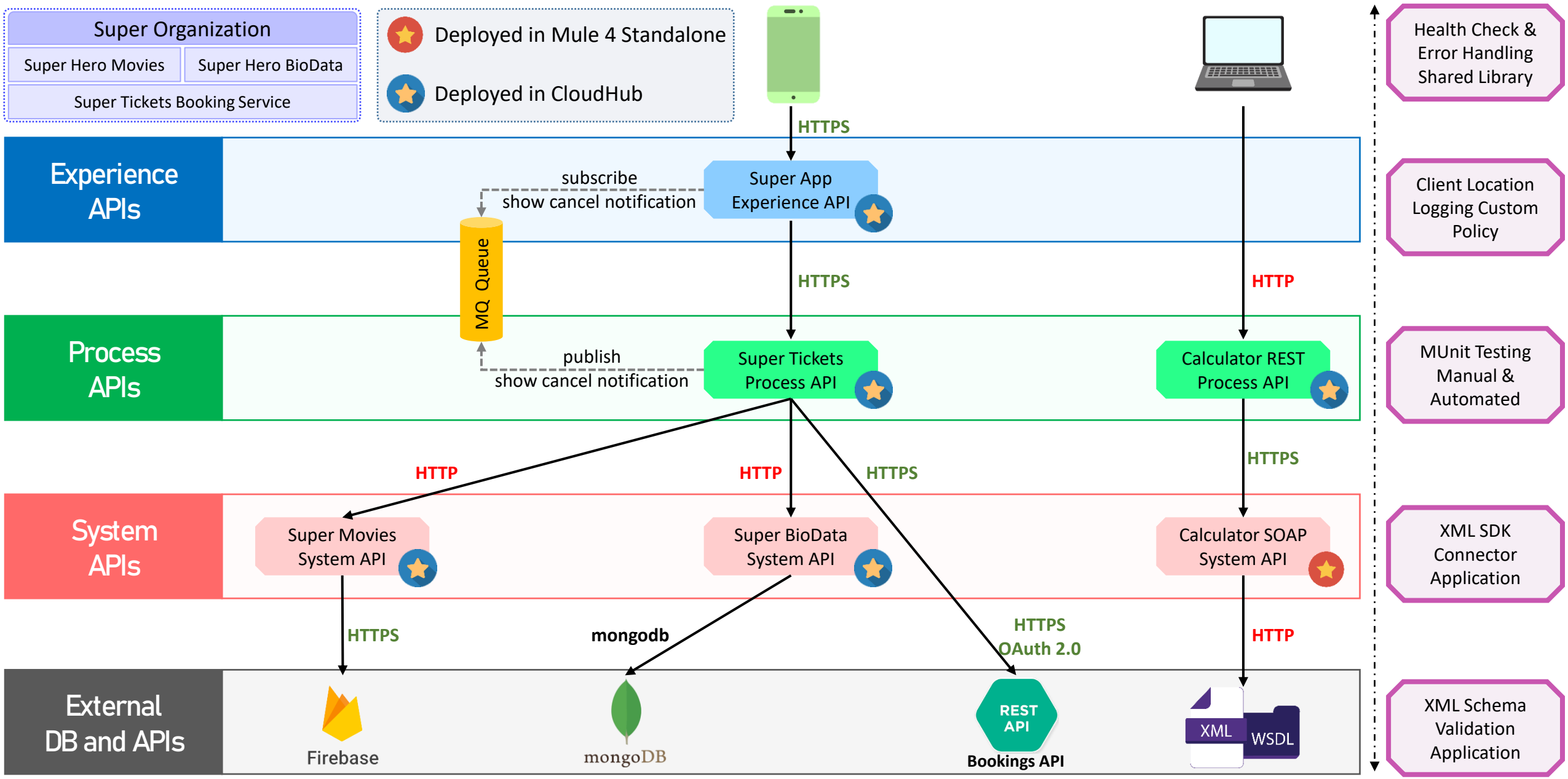
SoapUI

POSTMAN

Usecase Architecture

- API-led Architecture
- Other Artefacts used in course

API-led Architecture & Artefacts



0.2 Anypoint Platform

- Create a new Account
  - Assign Environments

0.3 API Specification

- Import RAML
- Publish RAML
- Explore Exchange

0.4 Anypoint Platform

- API Manager
  - Create API Instance
  - Add Automated Policy

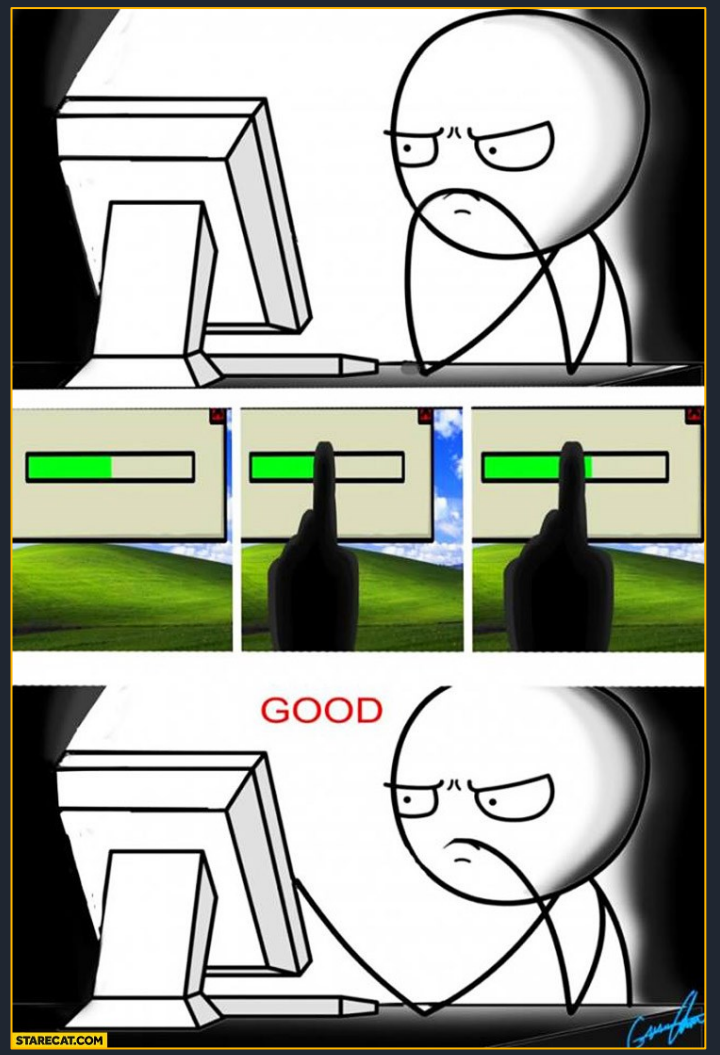
0.5 Anypoint Studio

- PART A
  - Set up Anypoint Studio
  - Local Maven Repo

- PART B
  - Import Mule Apps
    - Replace ORG ID
    - Replace API ID
    - Add MongoDB Driver
  - Deploy to CloudHub

0.6 Access Applications

- Add CH URL to API Instance
- Expose API to Public
- Get Credentials
- Access Apps

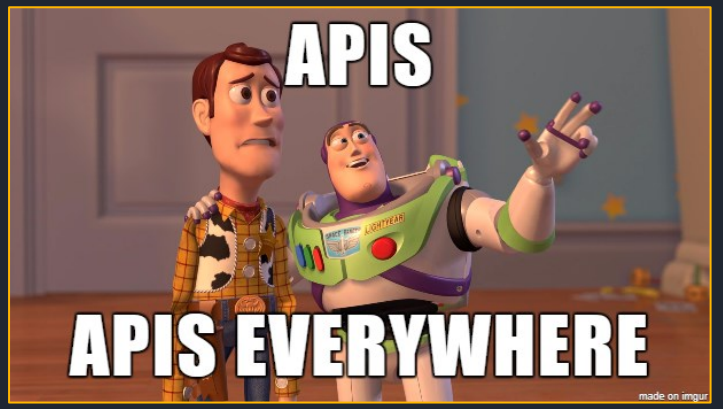


#1 Setting Project Structure, Deployment Strategy and Development Best Practises

- Add OAS to Design Center and Publish to Exchange
- Configure API Instance in API Manager
- Import API from Exchange to Anypoint Studio
- Configure API Autodiscovery
- Coding Conventions
- Secure Communication – Cryptography
- Self-signed Certification creation
- CloudHub Architecture
- Configuration Properties
- Secure Properties
- Hidden Properties
- Maven Resource Filtering
- Reducing Build Redundancy
- Maven Basic
- CloudHub Maven Deployment
- Anypoint Visualizer

API Lifecycle

- What/Why/How APIs
- API Lifecycle
- API Specification Options



API

CUSTOMER

APPLICATION

DEVELOPER

API

API TEAM

BACKEND

USERS

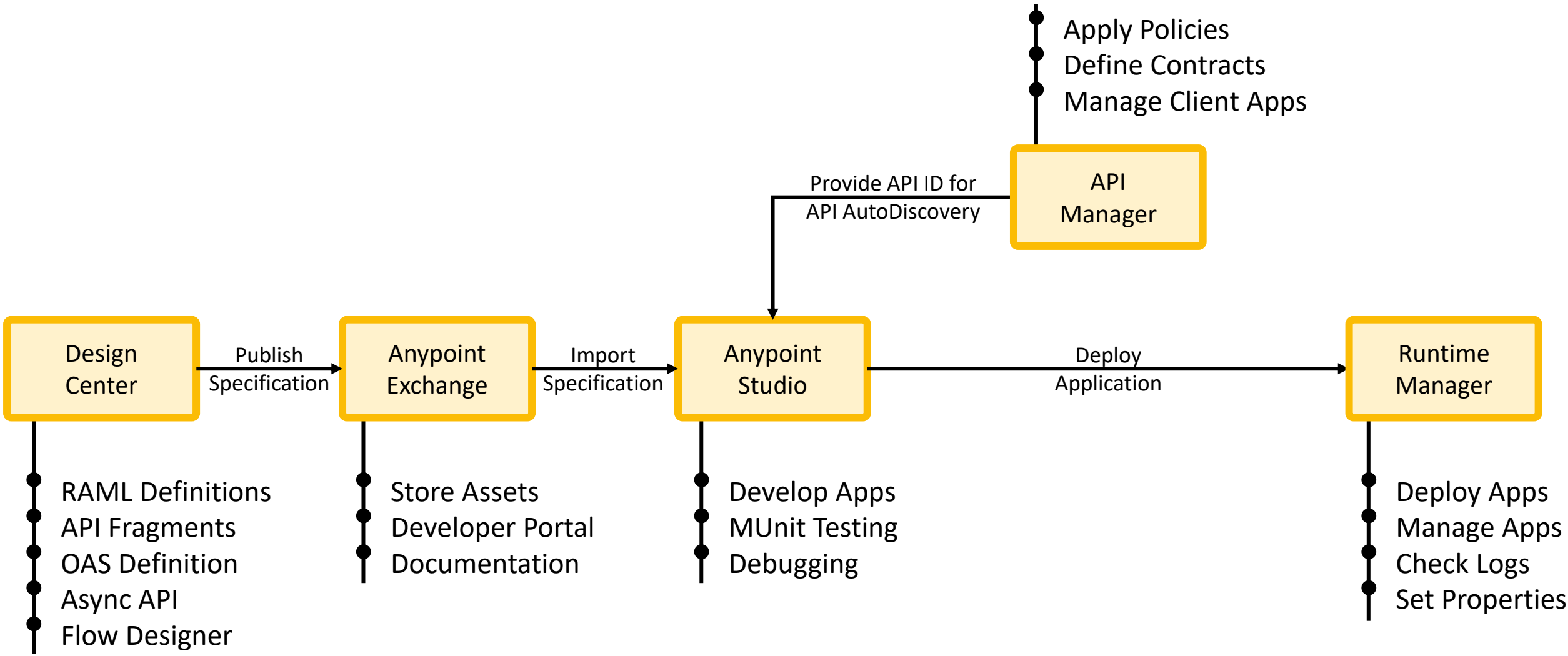
UBER APP

UBER APP DEVELOPERS

GOOGLE API DEVELOPERS

CLOUD BIGTABLE

API Lifecycle



API Specification

RAML

super-movies-sapi/master

```
#%RAML 1.0
title: Super Heros Movies Data SAPI
description: Super-Movies-SAPI allows you to fetch the basic movies o
heros.
version: 1.0
mediaType: application/json

types:
  movies-data: !include movies_dataType.raml

securitySchemes:
  basic-auth: !include exchange_modules/org.mule.examples/basic-auth-
basic-auth-securityscheme.raml

securedBy: basic-auth

/movies/{id}:
  uriParameters:
    id:
      required: true
      example:
        value: MARVEL0001
  get:
    description: This resources fetches bio data based on the unique *
Parameter
    responses:
      200:
        description: If **id** exists, returns the bio data
```

Open API Specification

Super Ticket PAPI/master

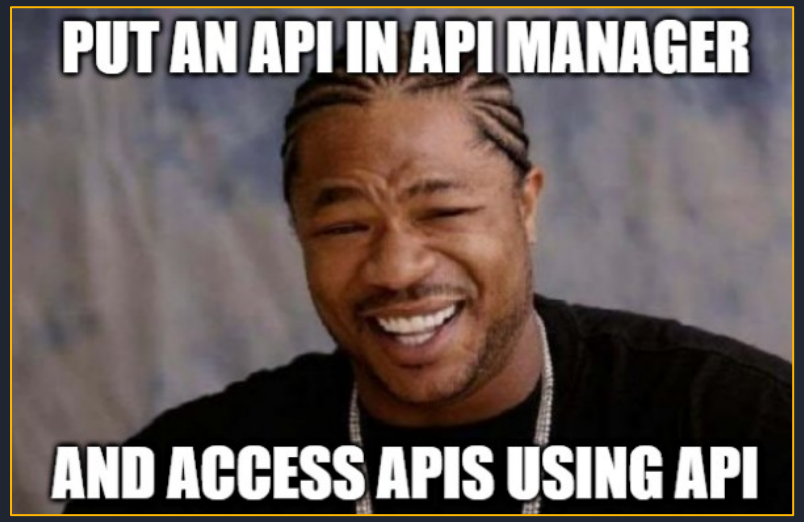
```
swagger: '2.0'
info:
  title: Super Ticket PAPI
  description: Super Ticket PAPI allows you to check the movie ticket
  version: v1
host: super-ticket-papi.us-e2.cloudhub.io
basePath: /api/v1
schemes:
  - https
paths:
  /tickets/{BID}:
    x-amf-description: Checks the passenger in with given number of ba
    get:
      parameters:
        - name: BID
          description: Unique Booking ID
          required: true
          in: path
          type: string
        - name: CID
          description: Unique Character ID
          required: false
          in: query
          type: string
      responses:
        '200':
          description: If the BID is correct, returns ticket booking d
Super Movies SAPI for the character data will be returned.
```

1.1 OAS Definition

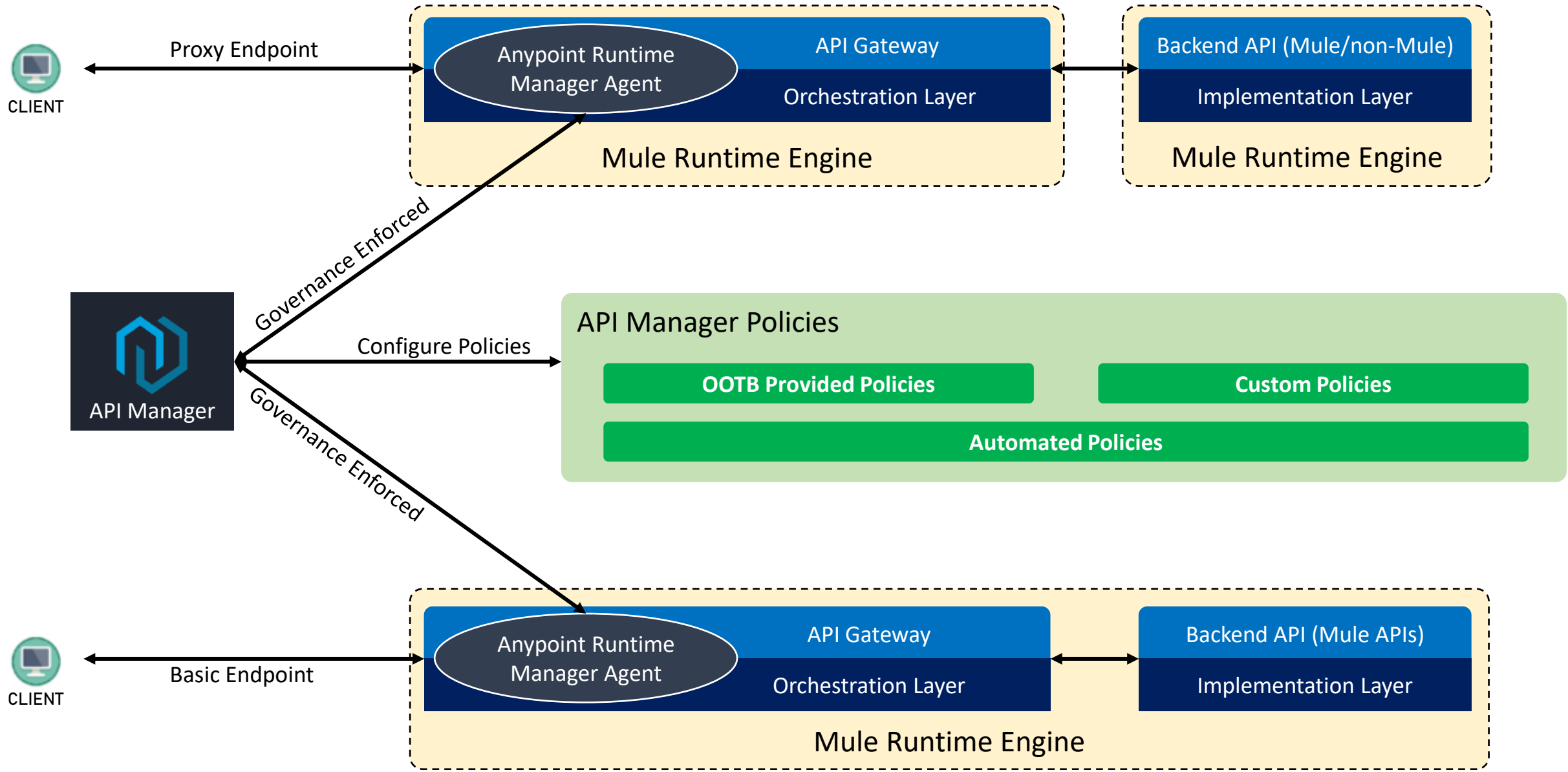
- Import OAS to Design Center
- Publish to Exchange

API Manager

- Proxy Endpoint
- Basic Endpoint
- API Gateway



API Manager & API Policies

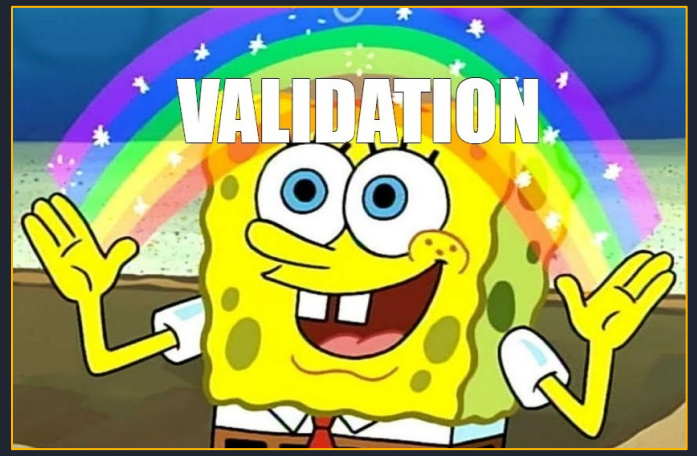


1.2 API Manager

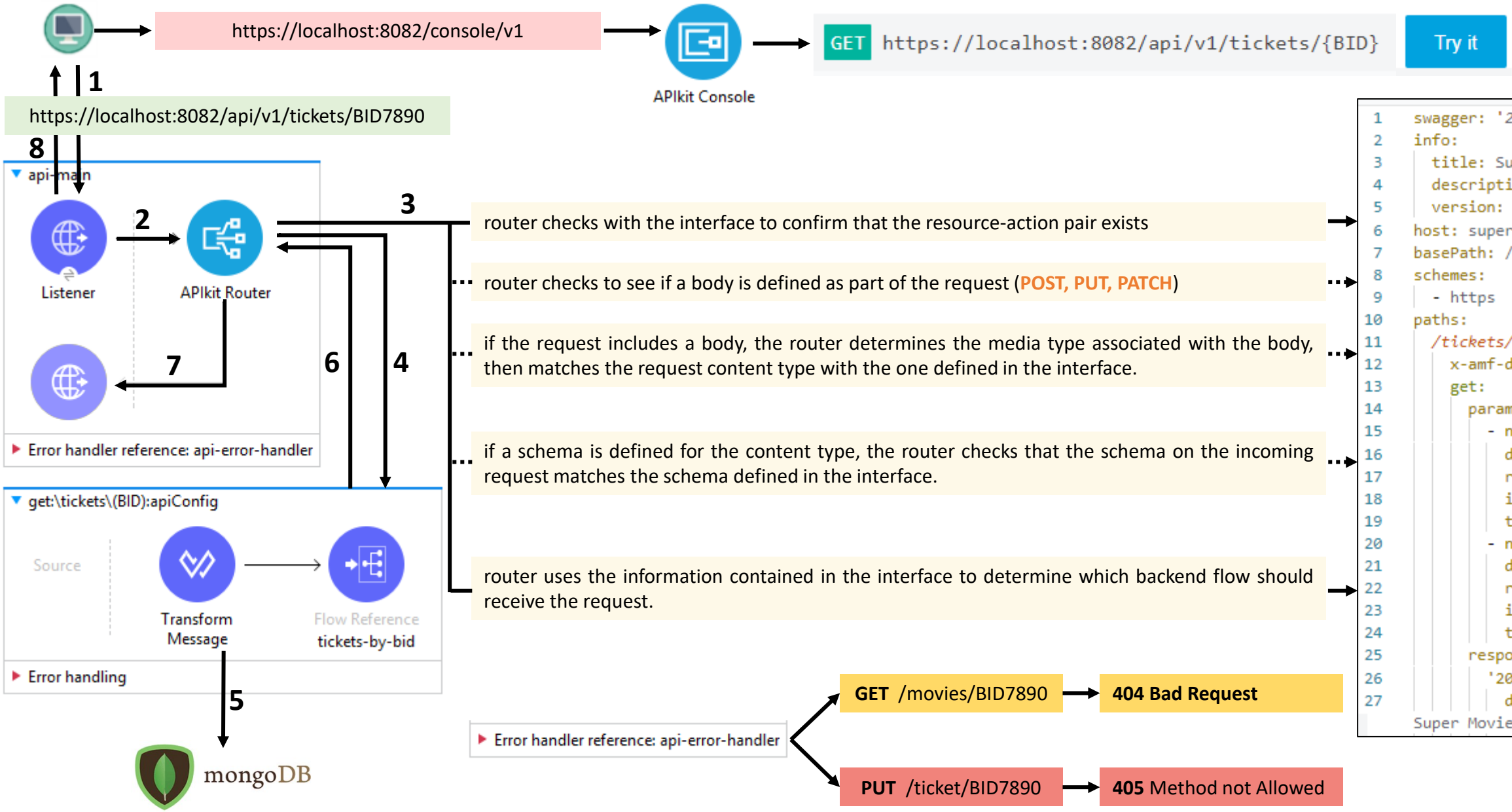
- Create API Instance
- Add Logging Policy

APKit Router

- api-main flow
- api-console flow
- error-handling
- How Routing works



APIKit Router

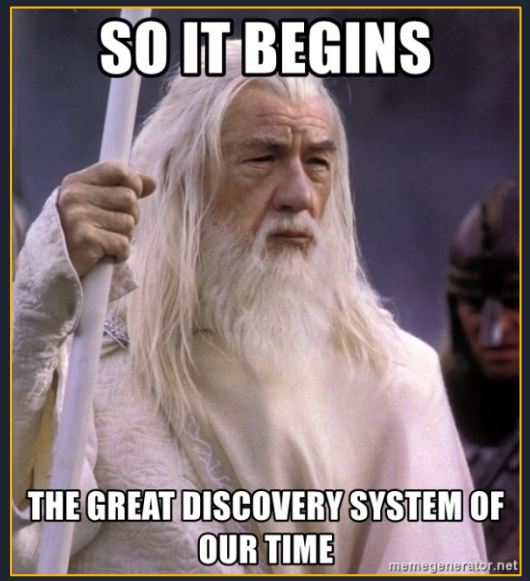


1.3 APIKit Router

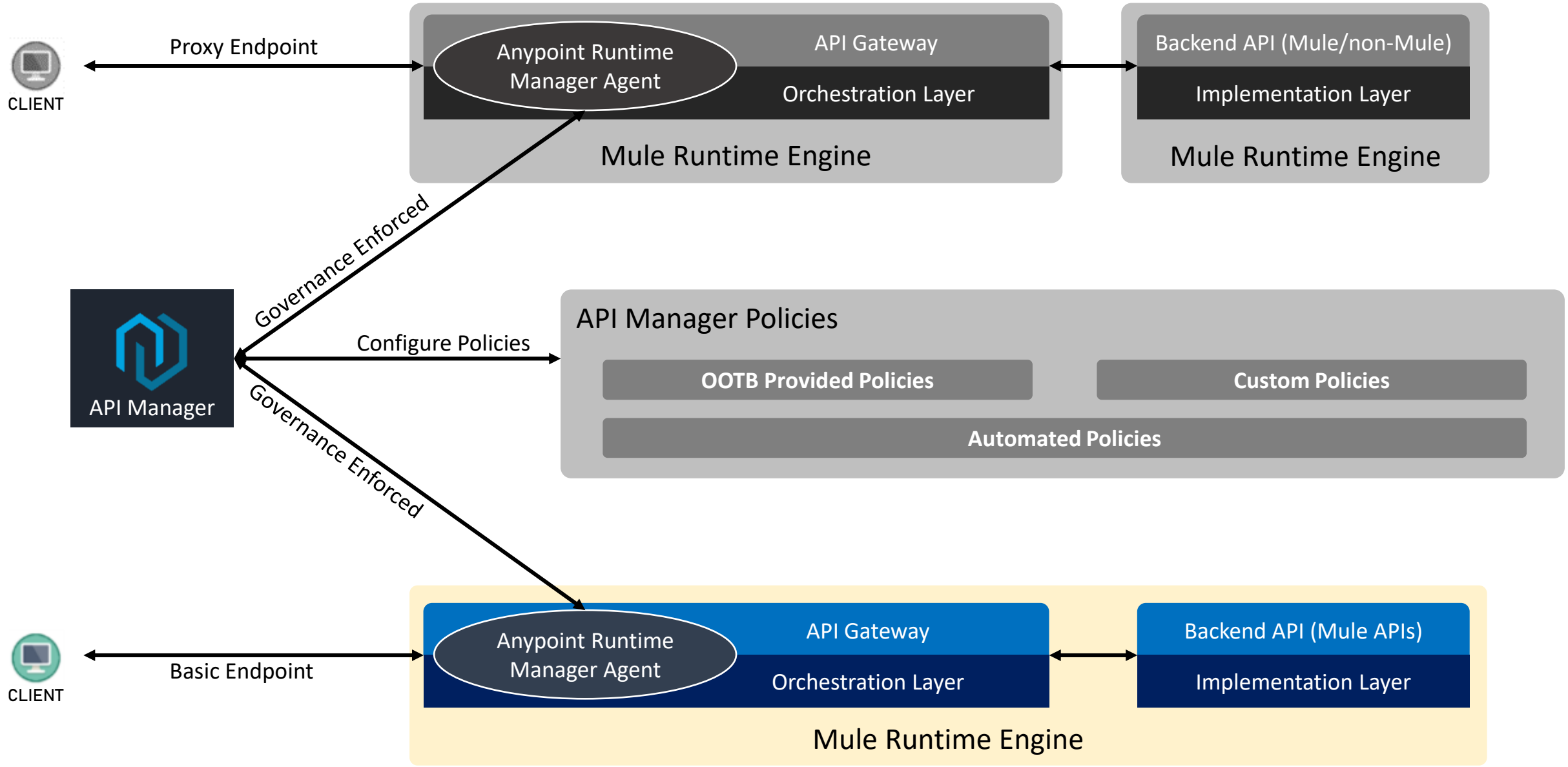
- Import API from Exchange to Studio
- Deploy the App

API Autodiscovery

- What is AutoDiscovery
- Steps to achieve it

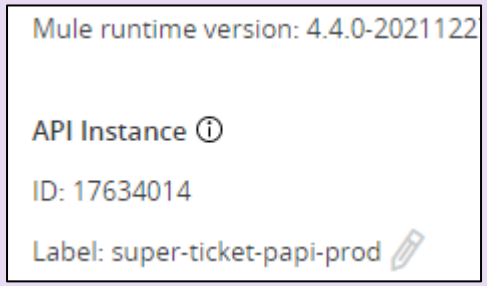


API Manager - Autodiscovery

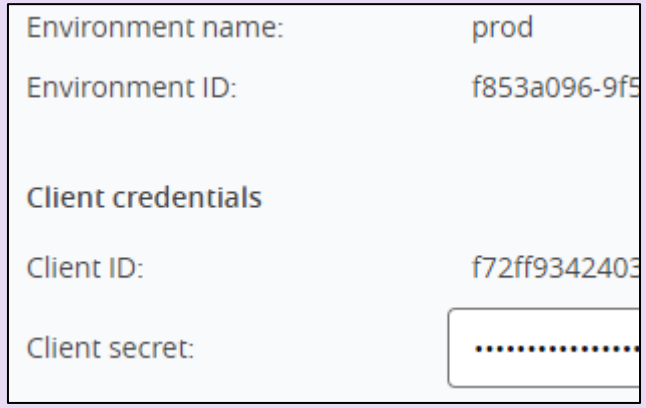


API Autodiscovery Steps

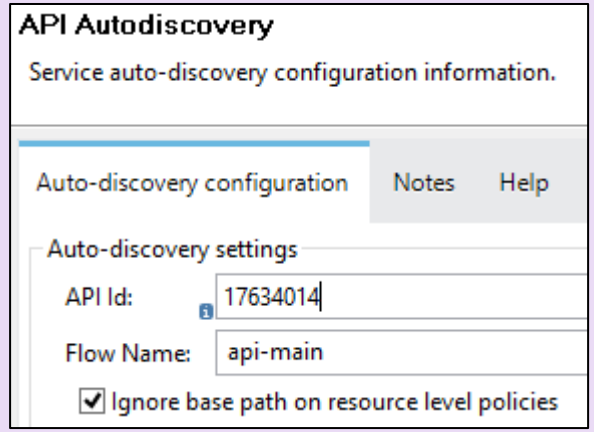
Get API Instance ID



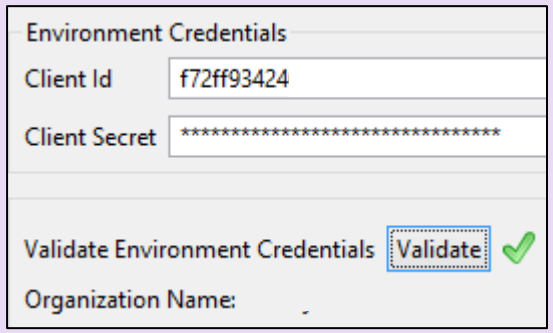
Get ENV's Client ID & Secret



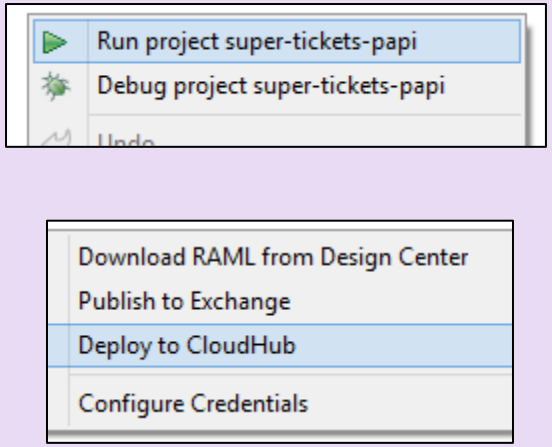
Add AAD Config in Studio



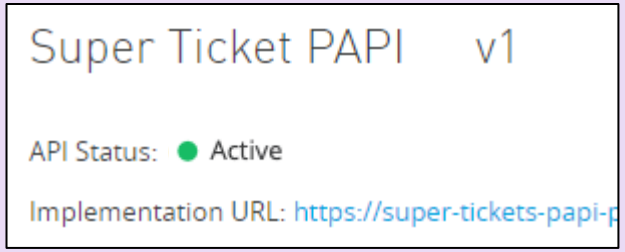
Add Env Credentials in Studio



Run local or Deploy CloudHub

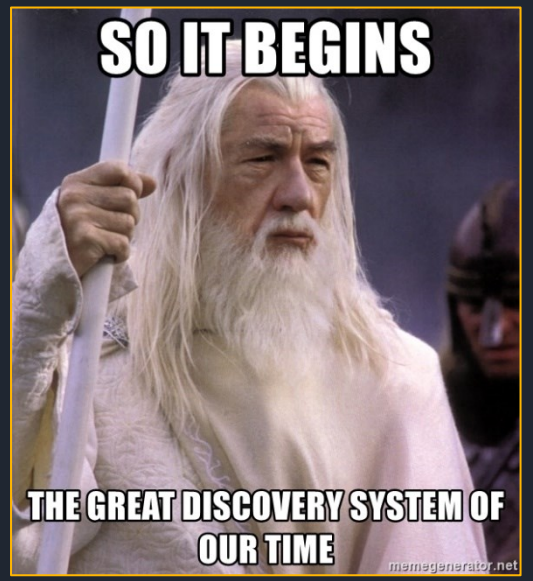


Check API Status



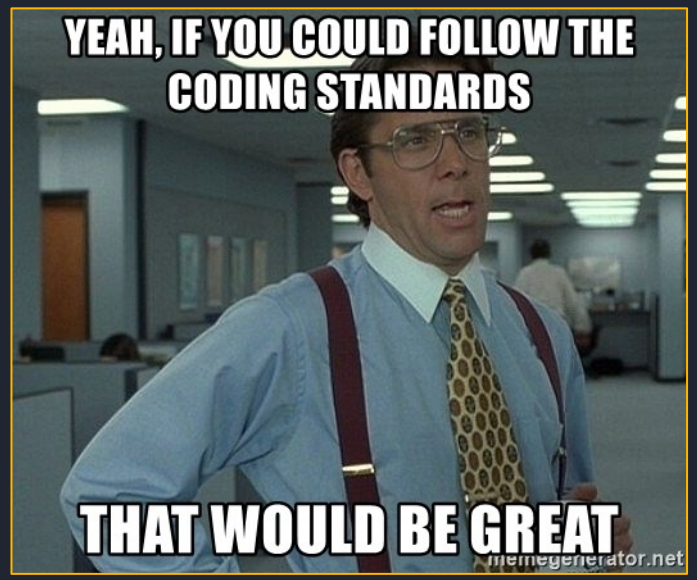
1.4 API Autodiscovery

- Steps to achieve it



Coding Conventions

- Naming Conventions
- Project Structure
- Anypoint Studio Settings



Coding Conventions

RAML Definition Name	Capitalize Word with Space	Super Movies SAPI	Super Tickets PAPI	Super Biodata SAPI
RAML Type/Library Name	Capitalize Word w/o Space	MoviesResponseType	TicketsResponseData	
Studio XML Formatter	Preferences	Split XML Attributes	Format Comments	Line Width
Mule App Project Name	kebab-case	super-tickets-papi	Same as Maven Artifact ID	
Mule Config File Names	lowercase	main.xml	api.xml	global.xml
Global Element Names	camelCase	apiHttpListernerConfig	oauthTokenObjectStore	anypointMQConfig
Flow | SubFlow Names	kebab-case	tickets-by-bid	compenstate-movies-service	get-biodata-by-cid-service
Flow Reference Name	Name should always be equal to referred/calling flow name			

Asset name

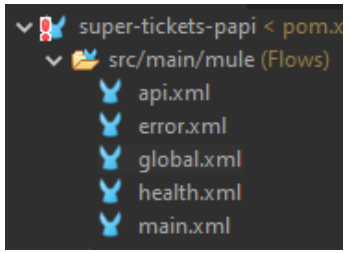
Super Ticket PAPI

Asset ID

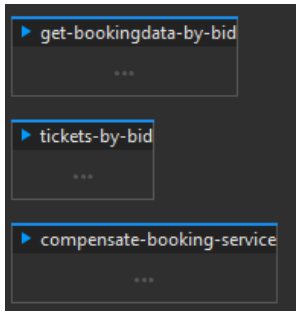
super-ticket-papi

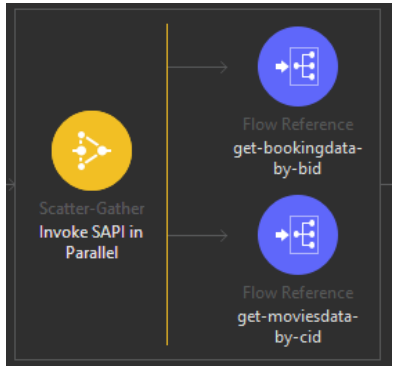
```
<vm:config
    name="vmConfig"
    doc:name="VM Config"
    doc:id="bab84e80-0082-45c4-87af-bf2864ec8302">
    <vm:queues>
        <vm:queue
            queueName="show-cancellation-notification-q"
            queueType="PERSISTENT"
            maxOutstandingMessages="100" />
```

```
<groupId>53a7ba06-a8d3-4536-9fb5-5c689add373f</groupId>
<artifactId>super-tickets-papi</artifactId>
<version>1.0.0</version>
```



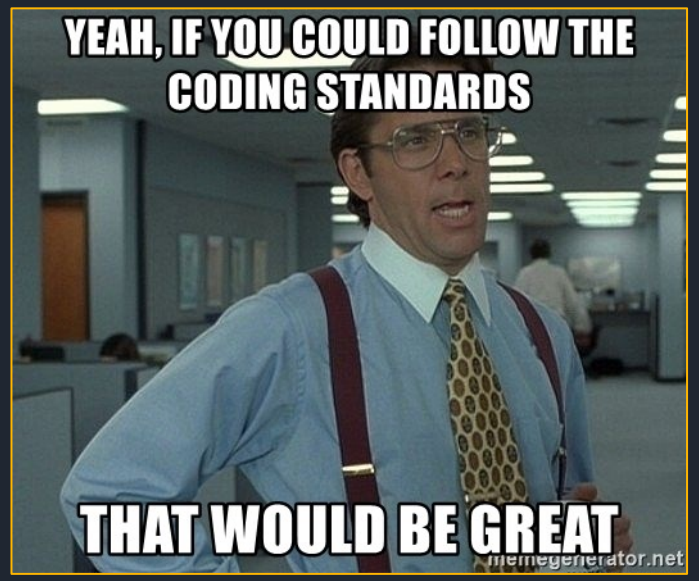
Type	Name
HTTP Listener config (Configuration)	apiHttpListernerConfig
Object store (Configuration)	oauthTokenObjectStore
Caching Strategy (Configuration)	bookingDataCachingStrategy





1.5 Coding Conventions

- Naming Conventions
- Project Structure
- Anypoint Studio Settings

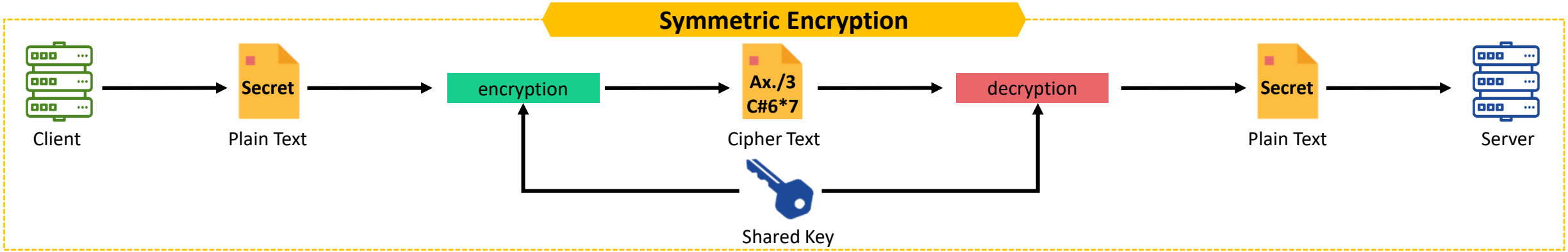


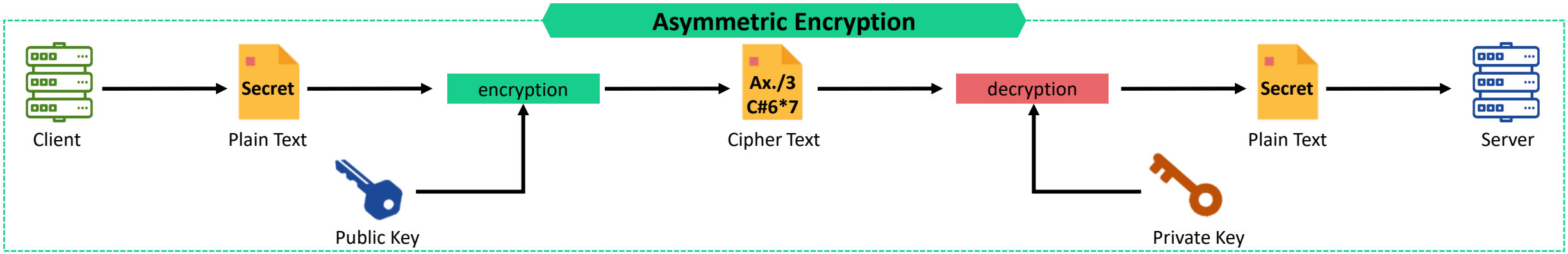
Secure Communication

- Using Cryptography
  - Symmetric Encryption
  - Asymmetric Encryption

Secure Communication – Using Cryptography

Symmetric Cryptography	Client and Server use the **same key** for encryption and decryption of messages		
	Same Key	Speed of encryption/decryption is Very **FAST**	Size of encrypted text is SAME or **LESS** than the original plaintext
Asymmetric Cryptography	Server issues a **Public Key** to the client to encrypt the messages | Server has a **Private Key** which is the **only key** to decrypt the message		
	Different Keys	Speed of encryption/decryption is **SLOWER**	Size of encrypted text is **MORE** than the original plaintext
Digital Certificates	Uses public/private key certificate signed by a **Trusted Authority (or self signed)** for communications		



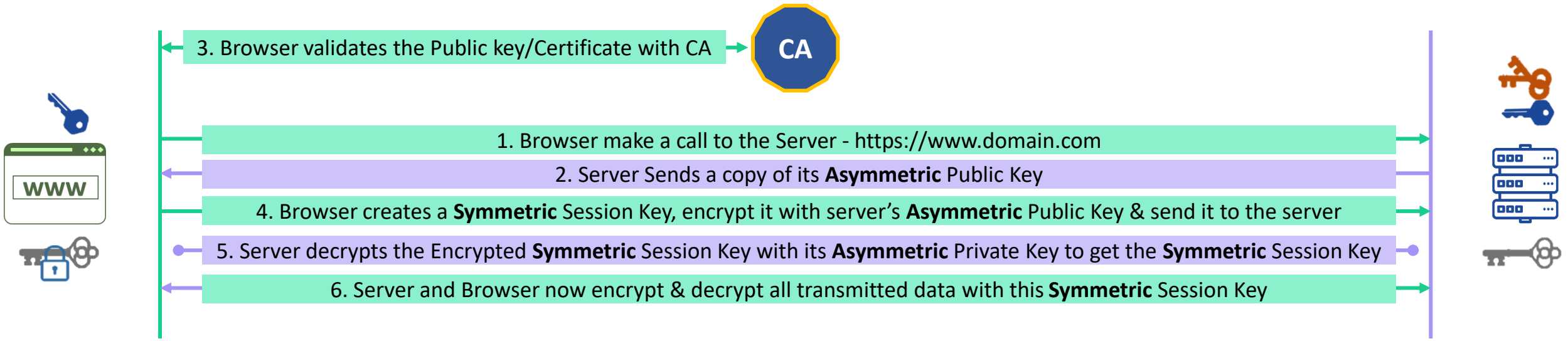


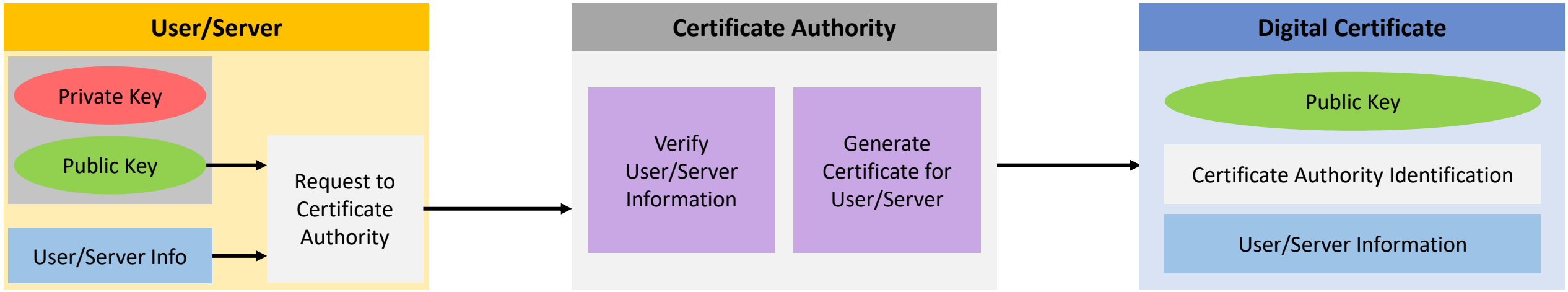
Secure Communication

- Digital Certificates
  - Certification Authority
  - Self-signed Certificate
  - Process Steps

Digital Certificates

Digital Certificates	Uses public/private key certificate signed by a **Trusted Authority (or self signed)** for communications
Certificate Authority	CA is an entity that validate the identities of entities and bind them to Public Keys to issue digital certificates *(ex - GoDaddy, Symantec)*
Self-Sign Certificate	A self-signed certificate is one that is not signed by a CA *(useful in test environments)*





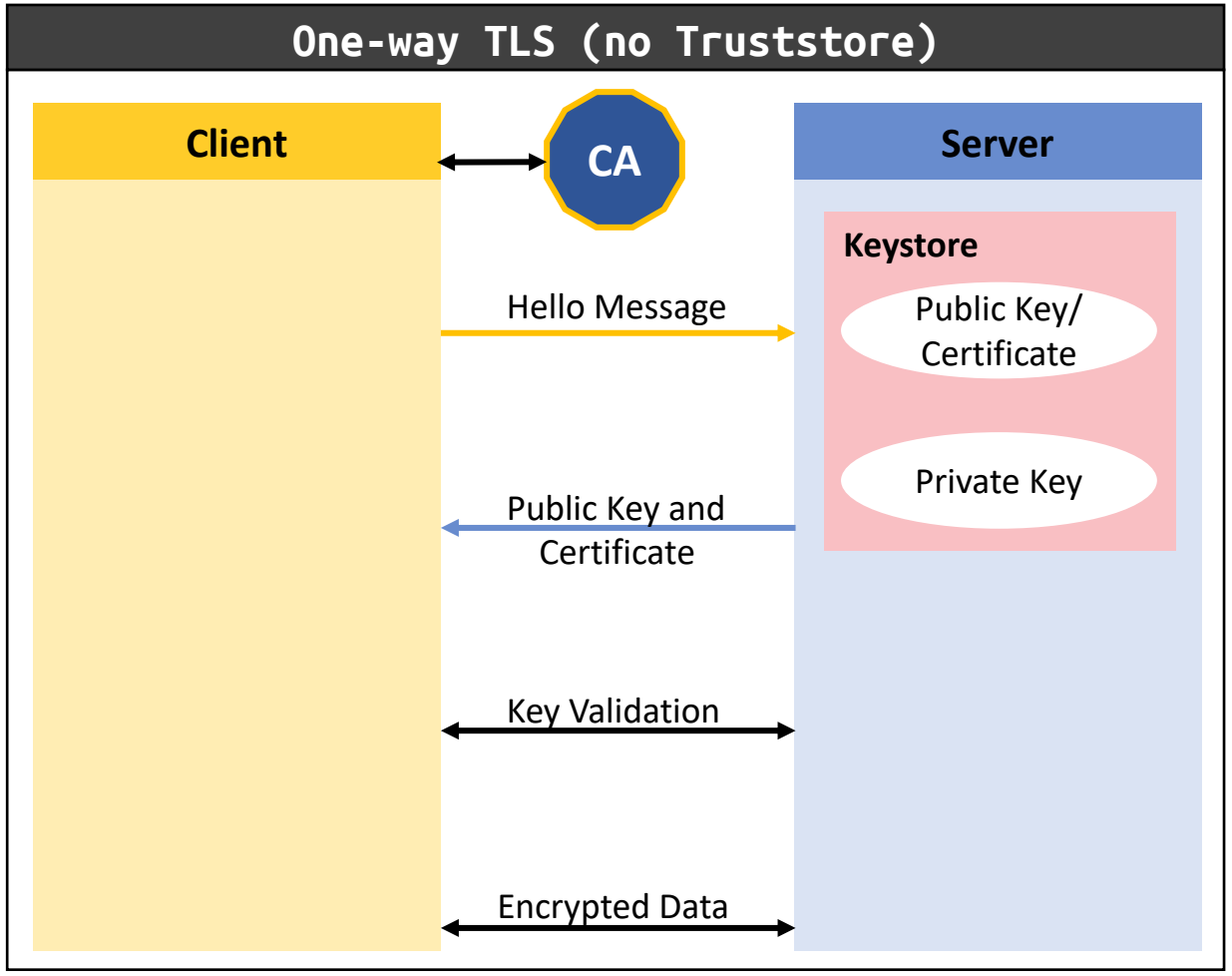
SSL/TLS Options

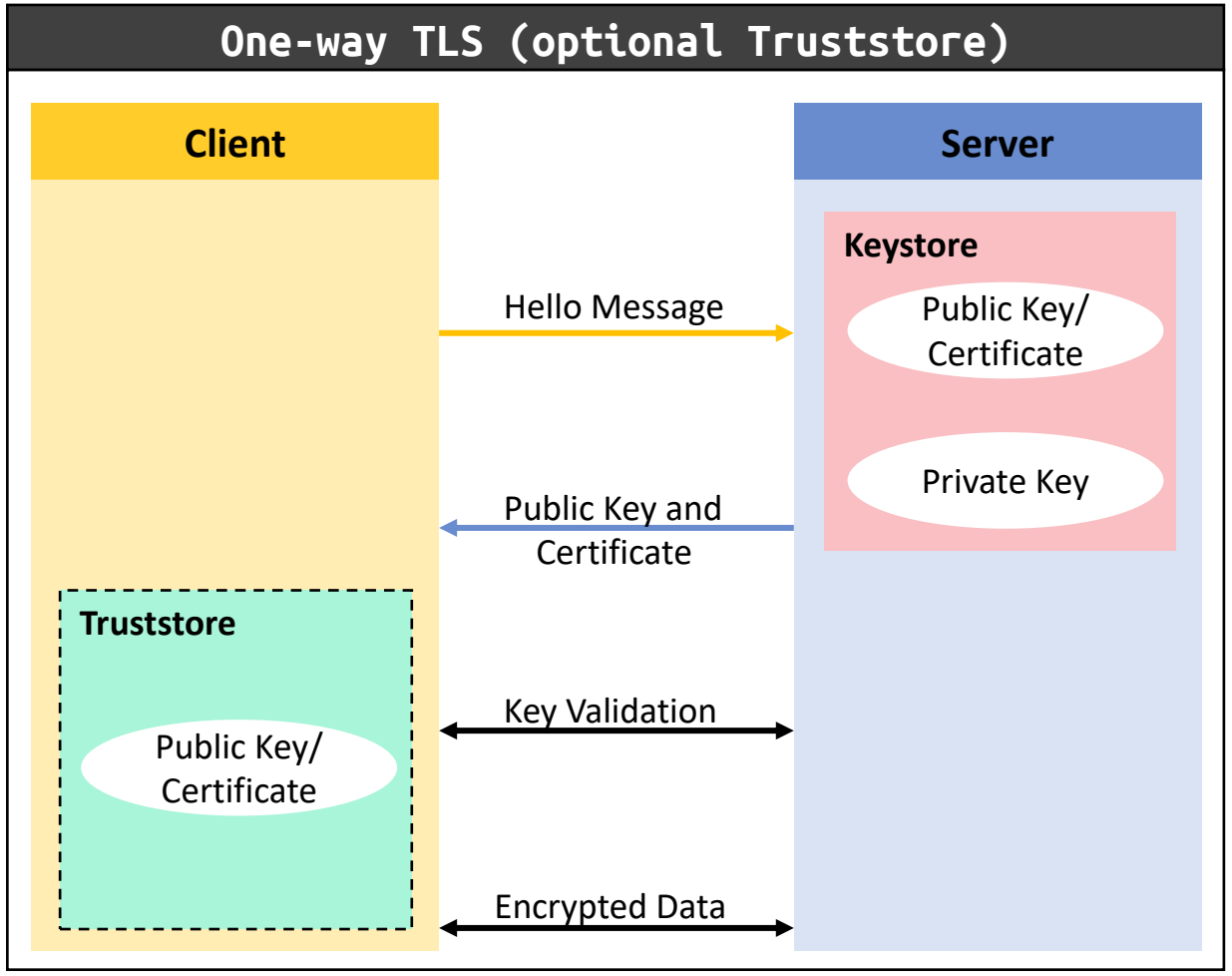
- One-way TLS
- Two-way TLS

One-way TLS

Keystore	Is a **repositories** that contains **Public certificates**, plus the corresponding **Private key** for clients or servers.
Truststore	Contains trusted certificates on a TLS client used to **validate** a TLS Server's Public certificate presented to the client *(typically self-signed certificates)*

In **One-way TLS** the **client** always **verifies** the **server** certificates and the **server** **NEVER** verifies the **client** certificates

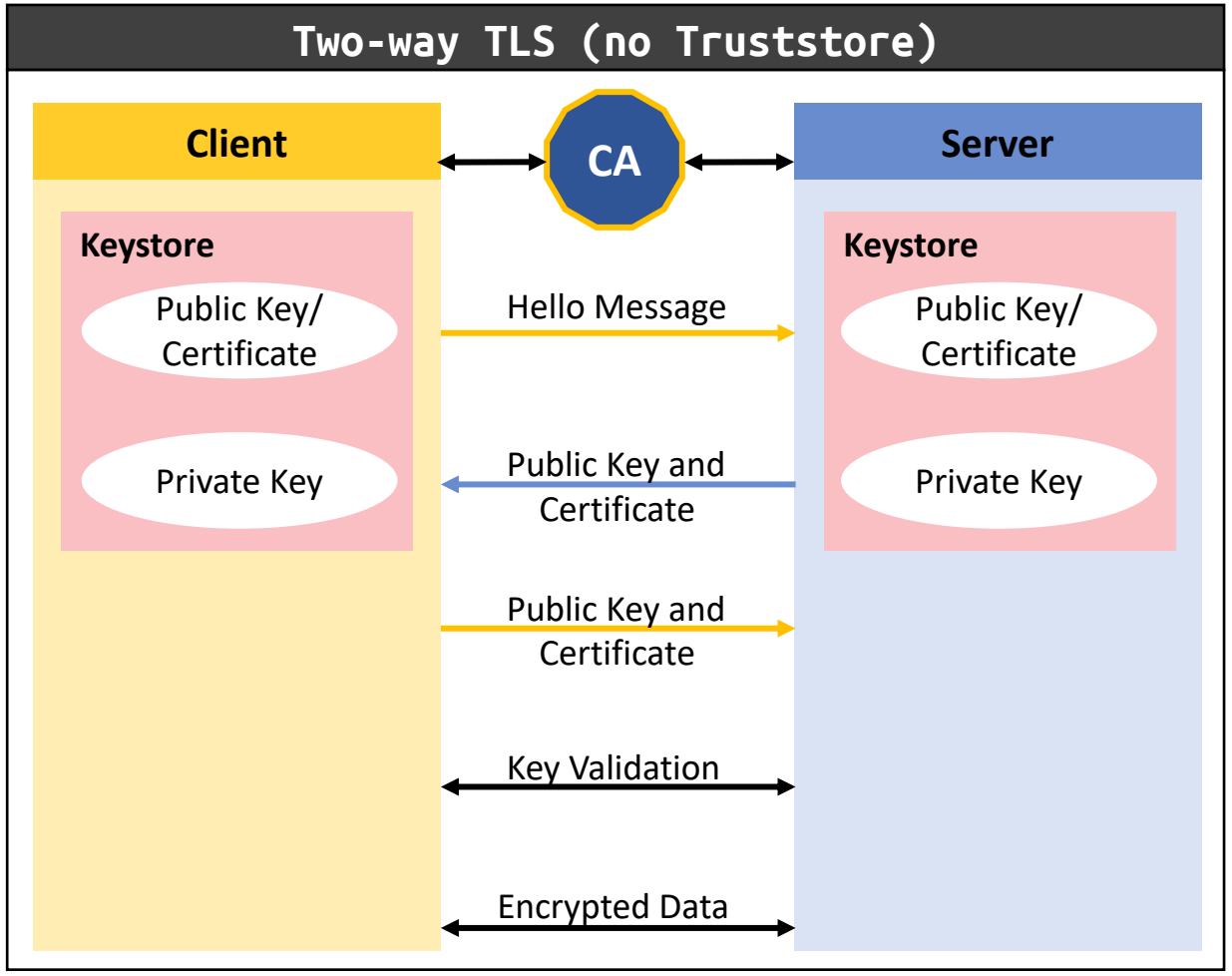


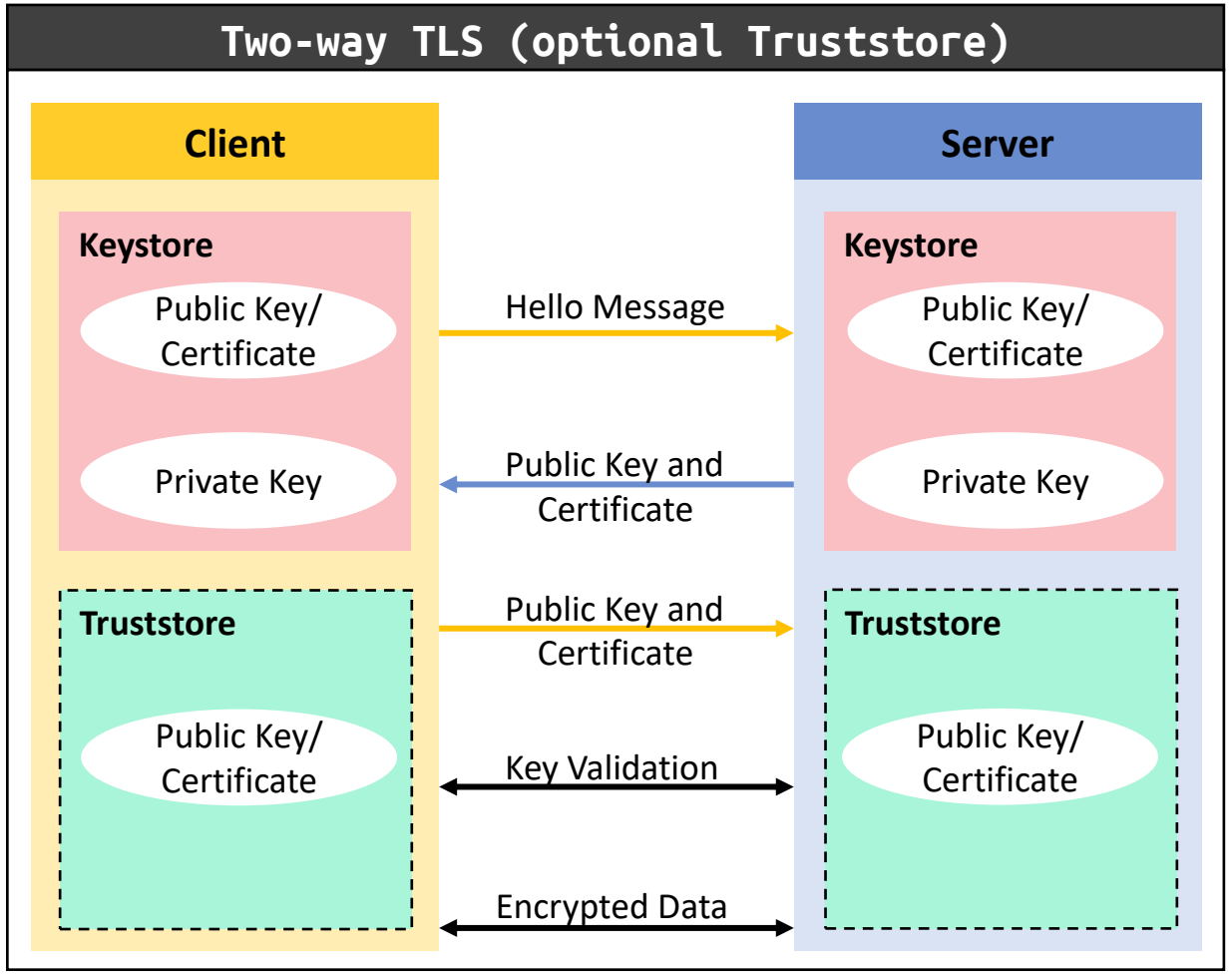


Two-way TLS

Keystore	Is a **repositories** that contains **Public certificates**, plus the corresponding **Private key** for clients or servers.
Truststore	Contains trusted certificates on a TLS client used to **validate** a TLS Server's Public certificate presented to the client *(typically self-signed certificates)*

In **Two-way TLS** the **client** verifies the **server** certificates and server **verifies** the client certificates.

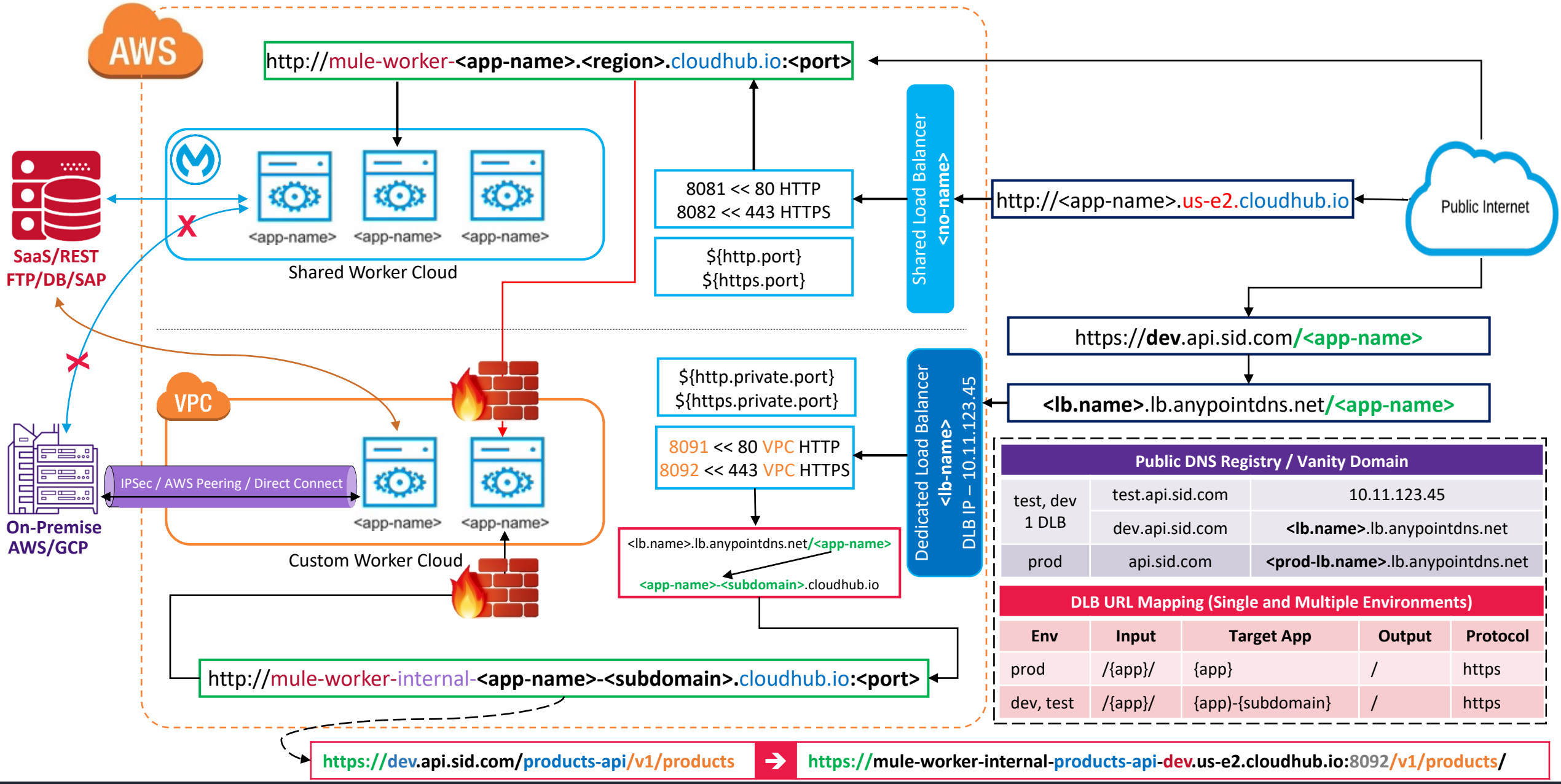




Architecture

- CloudHub Architecture

CloudHub Architecture



Public DNS Registry / Vanity Domain		
test, dev 1 DLB	test.api.sid.com	10.11.123.45
	dev.api.sid.com	**<lb.name>**.lb.anypointdns.net
prod	api.sid.com	**<prod-lb.name>**.lb.anypointdns.net

DLB URL Mapping (Single and Multiple Environments)				
Env	**Input**	**Target App**	**Output**	**Protocol**
prod	/{app}/	{app}	/	https
dev, test	/{app}/	{app}-{subdomain}	/	https

https://dev.api.sid.com/products-api/v1/products → https://mule-worker-internal-products-api-dev.us-e2.cloudhub.io:8092/v1/products/

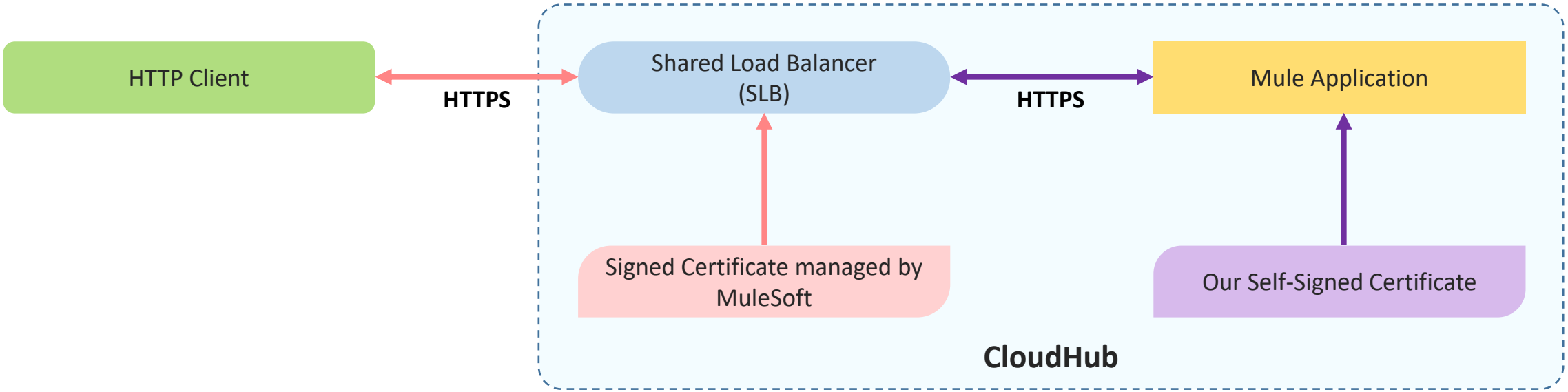
Self-Signed Cert

- Generate a Self-signed certificate
  - JKS vs PKCS12

Create Self Signed Certificate – One-Way-TLS

Create A Private/Public Key Pair - ***server-keystore.jks*** file contains the newly generated public and private key pair

```
keytool -genkey -alias mule-server -keysize 4096 -keyalg RSA -keystore server-keystore.jks
```



Create A Private/Public Key Pair - ***server-keystore.p12*** file contains the newly generated public and private key pair

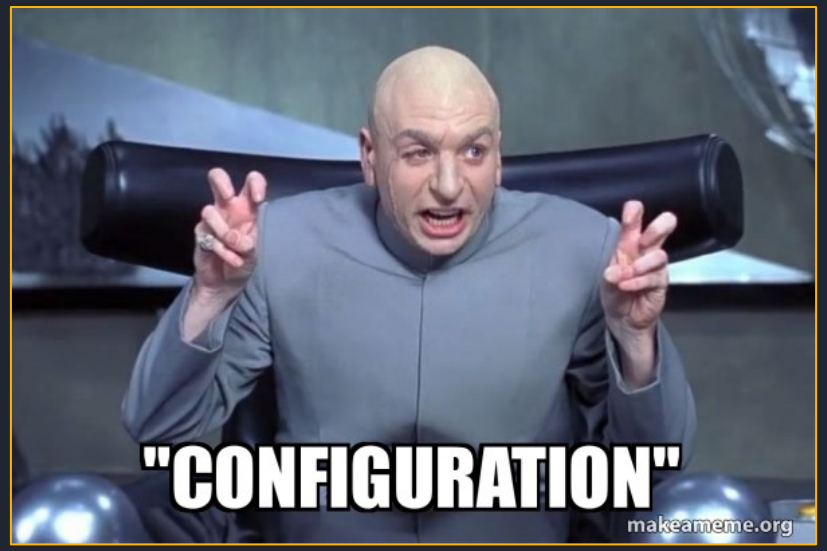
```
keytool -genkey -alias mule-server -keysize 2048 -keyalg RSA -keystore server-keystore.p12 -storetype pkcs12
```

1.6 Self-Signed Cert & CloudHub

- Generate a Self-signed certificate
- Configure it in Mule Application
- Run Locally
- Deploy to CH manually

Configuration Properties

- Types of property files
- Naming Convention
- Order of definitions



Configuration Values in Property Files

properties.yaml

```yaml
api:
  groupId: '${api.groupId}'
  artifactId: '${api.artifactId}'
  version: '${api.version}'
  majorVersion: 'v1'
  port: '8082'
  spec: 'resource::${api.groupId}:${api.
  id: '7777777'
```

properties-dev.yaml

```yaml
tls.keystore:
  type: 'pkcs12'
  path: 'certs/super-ticket-papi.p12'
  alias: 'server'

api:
  id: '1111111'
```

properties-prod.yaml

```yaml
tls.keystore:
  type: 'pkcs12'
  path: 'certs/prod-super-ticket-pa
  alias: 'server'

api:
  id: '4444444'
```

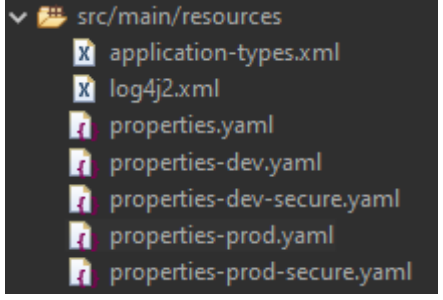
properties-dev-secure.yaml

```yaml
tls.keystore:
  keyPassword: '![GMdtU/4F1GrYHnFGdMmd6Yb8oAgPFuSp]'
  password: '![GMdtU/4F1GrYHnFGdMmd6Yb8oAgPFuSp]'
```

properties-prod-secure.yaml

```yaml
tls.keystore:
  keyPassword: '![GMdtU/4F1GrYHnFGdMmd6Yb8oAgPFuSp]'
  password: '![GMdtU/4F1GrYHnFGdMmd6Yb8oAgPFuSp]'
```

File Structure



global.xml >> TLS Configuration

```xml
<tls:context name="apiTLSContext">
    <tls:key-store
        type="pkcs12"
        path="${tls.keystore.path}"
        alias="${tls.keystore.alias}"
        keyPassword="${secure::tls.keystore.keyPassword}"
        password="${secure::tls.keystore.password}" />
```

global.xml >> Property Configurations

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

<secure-properties:config file="properties-${env}-secure.yaml" key="${secure.key}">
    <secure-properties:encrypt algorithm="Blowfish" />
</secure-properties:config>

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

`${api.id}` `${secure::tls.keystore.password}`

Mule 4 – Wrong Order of Definition

```xml
<configuration-properties   file="properties.yaml" />

<secure-properties:config file="properties-${env}-secure.yaml" key="${secure.key}">
    <secure-properties:encrypt algorithm="Blowfish" />
</secure-properties:config>

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

if `${env}` = **dev**

then `${api.id}` = **7777777**

Mule 4 – Correct Order of Definition

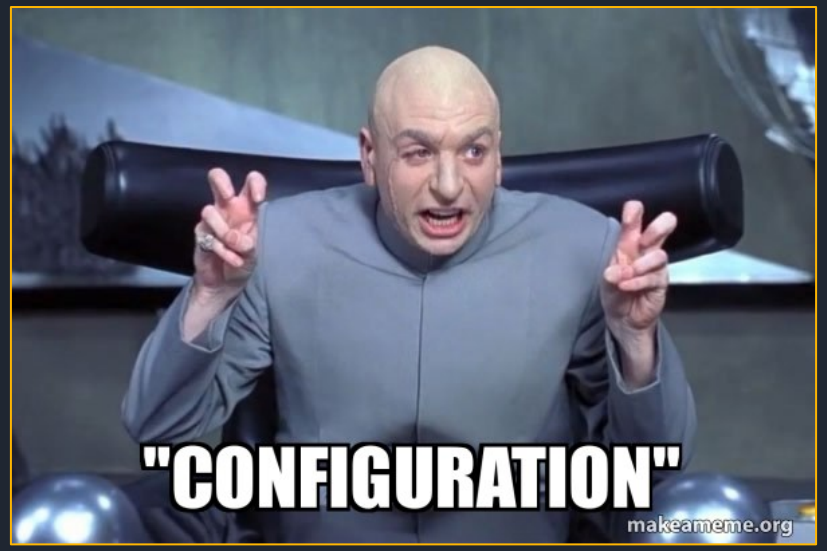
```xml
<configuration-properties file="properties-${env}.yaml" />

<secure-properties:config file="properties-${env}-secure.yaml" key="${secure.key}">
    <secure-properties:encrypt algorithm="Blowfish" />
</secure-properties:config>

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

if `${env}` = **dev**

then `${api.id}` = **1111111**

1.7 Configuration Properties

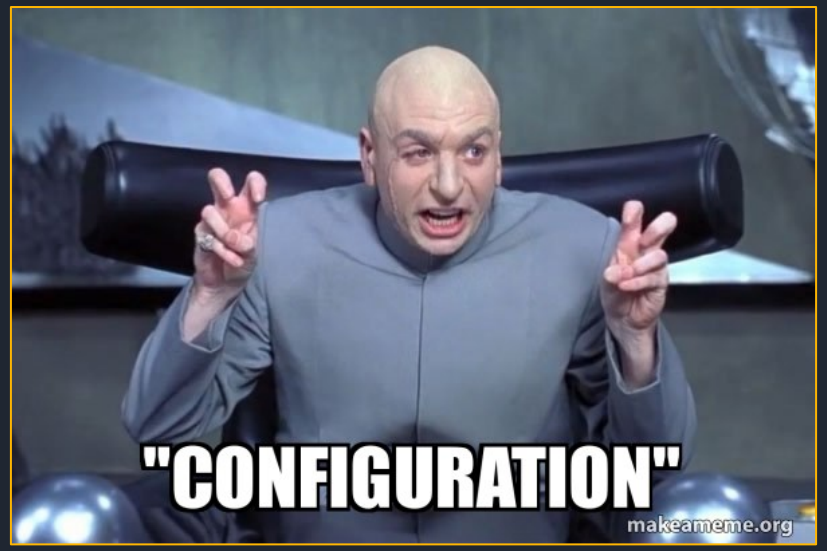
- Add properties.yaml



1.8 Configuration Properties

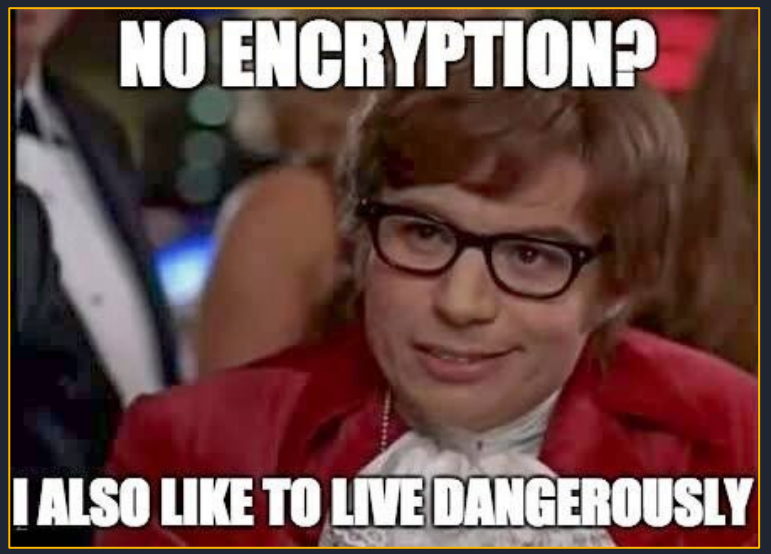
- Add environment specific property files
  - properties-dev.yaml
  - properties-prod.yaml

- Promote APIs to PROD env
  - Deploy to PROD



Secure & Safe Properties

- Encrypting properties
- Safe-Hidden properties



Secure and Safe Properties

Encryption by Admin/OpsTeam

```
tls.keystore:
  keyPassword: 'superSecretPassword'
  password: 'superSecretPassword'
```

```
java -cp secure-properties-tool.jar \
com.mulesoft.tools.SecurePropertiesTool \
<method> \
<operation> \
<algorithm> \
<mode> \
<key> \
<value>
```

```
java -cp secure-properties-tool.jar \
com.mulesoft.tools.SecurePropertiesTool \
string \
encrypt \
Blowfish \
CBC \
dontLeakThisKeySecureKey@7 \
superSecretPassword

GMdtU/4F1GrYHnFGdM
```

Details Sent to the Developers -

```
<algorithm>
<mode>
Encrypted value of Password
```

Configured in Mule App by Developer

```
tls.keystore:
  keyPassword: '![GMdtU/4F1GrYHnFGdM]'
  password: '![GMdtU/4F1GrYHnFGdM]'
```

```
<tls:key-store
...
keyPassword="${secure::tls.keystore.keyPassword}"
password="${secure::tls.keystore.password}" />
```

Mule Secure Configuration Property Extension

```
62    <secure-properties:config
63        file="properties-${env}-secure.yaml"
64        key="${secure.key}">
65        <secure-properties:encrypt
66            algorithm="Blowfish" />
67    </secure-properties:config>
```

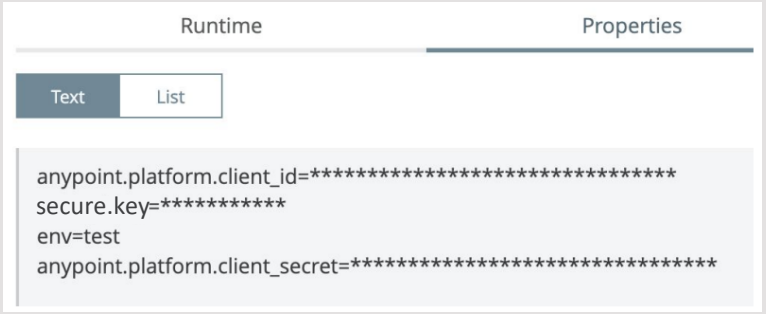
Add secure.key to mule-artifact.json

```
{
        "minMuleVersion": "4.4.0",
        "secureProperties": [
                "secure.key",
                "superSecretPassword"
                ]
}
```

App Deployed through CICD/Admin/OpsTeam

Deploy the app to CloudHub and add properties values either using CloudHub Properties Tab or thorugh a Maven cmd.

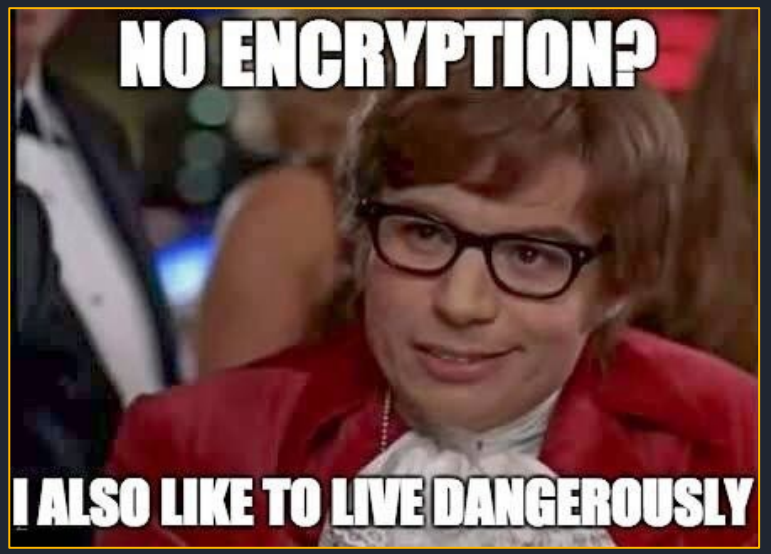
```
mvn deploy -DmuleDeploy \
-M-Dsecure.key=dontLeakThisKeySecureKey@7 \
-M-Danypoint.platform.client_id=4janfGJu..\
-M-Danypoint.platform.client_id=G9ONjgu2...
```



These values will be masked and can never be retrieved again via Runtime Manager GUI or API

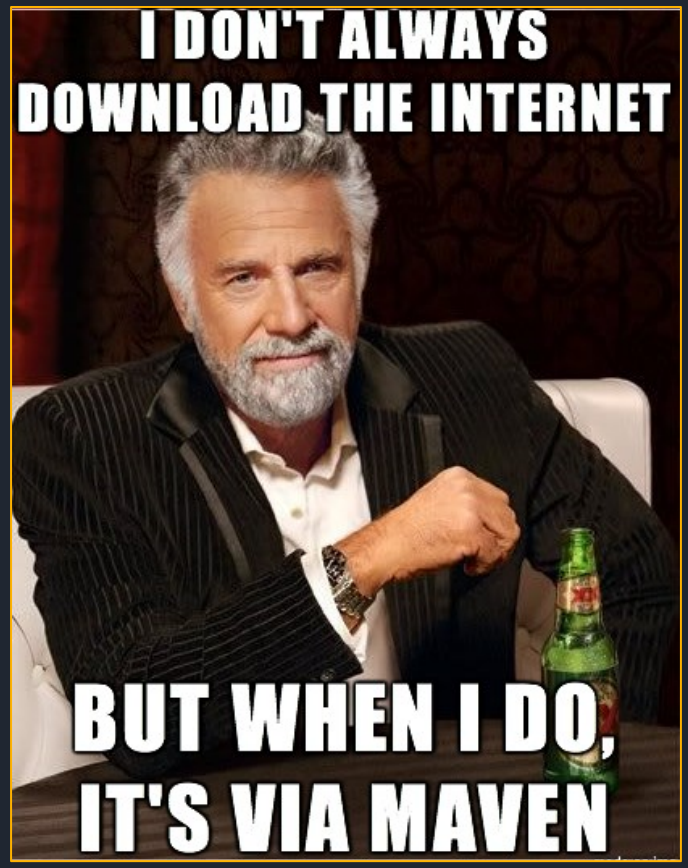
1.9 Secure & Safe Properties

- Encrypting properties
- Configuring them in app
- Safely hide secure keys
- Deploy the app



Maven

- Maven Basics



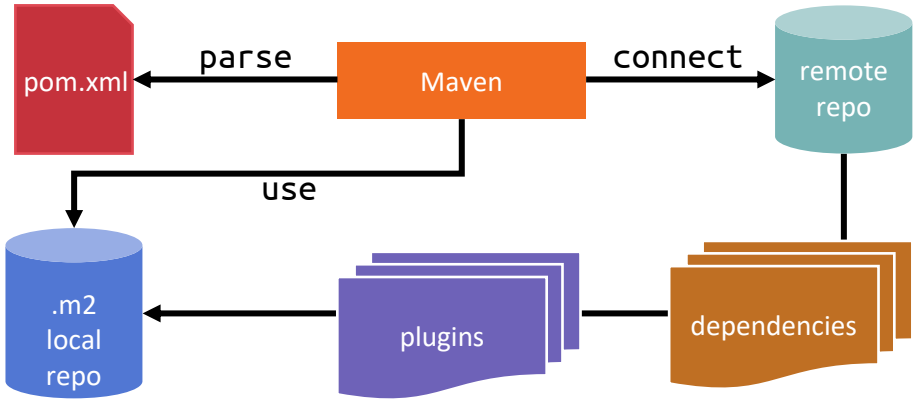
Maven

Maven Coordinates
groupId + artifactId + version

```xml
<project ...
    <groupId>53a7ba06-a8d3-4536-9fb5-5c689add373f</groupId>
    <artifactId>super-tickets-papi</artifactId>
    <version>1.0.0</version>
...
```

```xml
<dependency>
    <groupId>org.mule.connectors</groupId>
    <artifactId>mule-http-connector</artifactId>
    <version>1.6.0</version>
    <classifier>mule-plugin</classifier>
</dependency>
```

```xml
<plugin>
    <groupId>org.mule.tools.maven</groupId>
    <artifactId>mule-maven-plugin</artifactId>
    <version>3.5.2</version>
    <extensions>true</extensions>
</plugin>
```



Lifecycle
clean
default
site

Phases
validate
initialize
compile
test
package
verify
install
deploy

Goals
compile:complie
surefire:test
jar:jar
install:install
deploy:deploy

mvn phase is same as **mvn plugin:goal**

settings.xml (Maven)

```xml
<settings>
  <servers>
    <server>
      <id>anypoint-exchange-v3</id>
      <username>~~~Client~~~</username>
      <password>c8e4e6e79?f9f10E41f</password>
    </server>
  </servers>
</settings>
```

Mule Maven Plugin			
	mvn org.mule.tools.maven:mule-maven-plugin:3.6.2:package	mvn package	<artifactId>-<version>-mule-application.jar
	mvn org.mule.tools.maven:mule-maven-plugin:3.6.2:deploy	mvn deploy	deploy Mule App JAR to CloudHub, RTF, Standalone

Resource Filtering

- What?
- Why?
- How?

Removing Configuration Redundancy

properties.yaml

```yaml
api:
  groupId: '53a7ba06-a8d3-4536-9fb5-5c689add373f'
  artifactId: 'super-tickets-papi'
  version: '1.0.0'
```

pom.xml

```xml
<dependency>
    <groupId>53a7ba06-a8d3-4536-9fb5-5c689add373f</groupId>
    <artifactId>super-ticket-papi</artifactId>
    <version>1.0.0</version>
    <classifier>oas</classifier>
    <type>zip</type>
</dependency>
```

pom.xml

```xml
<properties>
    <api.groupId>53a7ba06-a8d3-4536-9fb5-5c689add373f</api.groupId>
    <api.artifactId>super-ticket-papi</api.artifactId>
    <api.version>1.0.0</api.version>
</properties>
```

pom.xml

```xml
<dependency>
    <groupId>${api.groupId}</groupId>
    <artifactId>${api.artifactId}</artifactId>
    <version>${api.version}</version>
    <classifier>oas</classifier>
    <type>zip</type>
</dependency>
```

properties.yaml

```yaml
api:
  groupId: '${api.groupId}'
  artifactId: '${api.artifactId}'
  version: '${api.version}'
```

pom.xml

```xml
<resources>
    <resource>
        <directory>src/main/resources</directory>
        <filtering>true</filtering>
    </resource>
</resources>
<testResources>
    <testResource>
        <directory>src/test/resources</directory>
        <filtering>true</filtering>
    </testResource>
</testResources>
<plugins>
    <plugin>
        <groupId>org.apache.maven.plugins</groupId>
        <artifactId>maven-resources-plugin</artifactId>
        <version>3.2.0</version>
        <configuration>
            <nonFilteredFileExtensions>
                <nonFilteredFileExtension>p12</nonFilteredFileExtension>
                <nonFilteredFileExtension>crt</nonFilteredFileExtension>
                <nonFilteredFileExtension>pem</nonFilteredFileExtension>
            </nonFilteredFileExtensions>
        </configuration>
    </plugin>
</plugins>
```

Maven Resource Filtering

1.10 Resource Filtering

- Configure Plugin
  - Include property file
  - Exclude certain Extensions
- Check /target directory

Build Redundancies & Duplication

- Dependency Management
- POM
  - Parent POM
  - Master BOM

Dependency & Plugin Management

pom.xml (John - Dev - Project_10)

```
57  ...
58  <plugin>
59    <groupId>org.mule.tools.maven</groupId>
60    <artifactId>mule-maven-plugin</artifactId>
61    <version>1.1.1</version>
62    <extensions>true</extensions>
63    <configuration>
64      <cloudHubDeployment>
65        <businessGroup />
66        <environment>dev</environment>
67        <region>us-east-2</region>
68        <muleVersion>4.4.0</muleVersion>
69        <workers>1</workers>
70        <workerType>MICRO</workerType>
71          ...
72  </plugin>
73  <dependencies>
74    <dependency>
75      <groupId>org.mule.connectors</groupId>
76        <artifactId>mule-http-connector</artifactId>
77        <version>1.1.1</version>
78        <classifier>mule-plugin</classifier>
79    </dependency>
80  ...
```

pom.xml (Sara - Dev - Project_10)

```
83  ...
84  <plugin>
85    <groupId>org.mule.tools.maven</groupId>
86    <artifactId>mule-maven-plugin</artifactId>
87    <version>2.2.2</version>
88    <extensions>true</extensions>
89    <configuration>
90      <cloudHubDeployment>
91        <businessGroup />
92        <environment>dev</environment>
93        <region>us-east-2</region>
94        <muleVersion>4.4.0</muleVersion>
95        <workers>2</workers>
96        <workerType>LARGE</workerType>
97          ...
98  </plugin>
99  <dependencies>
100   <dependency>
101     <groupId>org.mule.connectors</groupId>
102       <artifactId>mule-http-connector</artifactId>
103       <version>1.2.2</version>
104       <classifier>mule-plugin</classifier>
105   </dependency>
106 ...
```

pom.xml (Sid - Dev - Project_15)

```
68  <plugin>
69    <groupId>org.mule.tools.maven</groupId>
70    <artifactId>mule-maven-plugin</artifactId>
71    <version>7.7.7</version>
72    <extensions>true</extensions>
73    <configuration>
74      <cloudHubDeployment>
75        <businessGroup />
76        <environment>dev</environment>
77        <region>us-east-2</region>
78        <muleVersion>4.4.0</muleVersion>
79        <workers>4</workers>
80        <workerType>4XLARGE</workerType>
81          ...
82  </plugin>
83  <dependencies>
84    <dependency>
85      <groupId>org.mule.connectors</groupId>
86        <artifactId>mule-http-connector</artifactId>
87        <version>7.7.7</version>
88        <classifier>mule-plugin</classifier>
89    </dependency>
90  ...
```

pom.xml (Mule Application POM)	→	pom.xml (Parent POM)	→	pom.xml (BOM - Bill of Material)
Required Dependencies & Plugins		Plugin Configurations		Dependencies & Plugin Versions

POM << Parent-POM << BOM

MCD2

../pom.xml (Mule Application POM)

```
148 ...
149 <parent>
150     <groupId>53a7ba06-a8d3-4536-9fb5</groupId>
151     <artifactId>parent-pom</artifactId>
152     <version>1.0.0</version>
153     <relativePath>../parent-pom/pom.xml</relativePath>
154 </parent>
155 ...
156
157 <plugin>
158     <groupId>org.mule.tools.maven</groupId>
159     <artifactId>mule-maven-plugin</artifactId>
160     <extensions>true</extensions>
161 </plugin>
162 ...
163
164 <dependency>
165     <groupId>org.mule.connectors</groupId>
166     <artifactId>mule-http-connector</artifactId>
167     <classifier>mule-plugin</classifier>
168 </dependency>
169 ...
```

../parent-pom/pom.xml (Parent POM)

```
1  <parent>
2      <groupId>53a7ba06-a8d3-4536-9fb5</groupId>
3      <artifactId>bom</artifactId>
4      <version>1.0.0</version>
5      <relativePath>../bom/pom.xml</relativePath>
6  </parent>
7  ...
8  <properties>
9      <type>custom</type>
10     <deployment.prefix>sid-</deployment.prefix>
11     <deployment.suffix>-${deployment.env}</deployment.su
12     <deployment.name>${deployment.prefix}....</deployment
13     <ap.environment>${deployment.env}</ap.environment>
14 </properties>
15 ...
16 <pluginManagement>
17   <plugins>
18     <plugin>
19       <groupId>org.mule.tools.maven</groupId>
20       <artifactId>mule-maven-plugin</artifactId>
21       <extensions>true</extensions>
22       <configuration>
23         <cloudHubDeployment>
24           <environment>${ap.environment}</environment>
25           <region>us-east-2</region>
26           <workers>1</workers>
27           <workerType>MICRO</workerType>
28           <applicationName>${deployment.name}</applicati
29           <properties>
30               <env>${deployment.env}</env>
31           </properties>
32         </cloudHubDeployment>
33       </configuration>
34     </plugin>
35   </plugins>
36 </pluginManagement>
37 ...
```

../bom/pom.xml (BOM)

```
1  ...
2  <properties>
3      <app.runtime.semver>4.4.0</app.runtime.semver>
4      <mule.maven.plugin.version>3.5.3</mule.maven.plugin.version>
5      <http.connector.version>1.6.0</http.connector.version>
6  </properties>
7  ...
8  <pluginManagement>
9      <plugins>
10         <plugin>
11             <groupId>org.mule.tools.maven</groupId>
12             <artifactId>mule-maven-plugin</artifactId>
13             <version>${mule.maven.plugin.version}</version>
14             <extensions>true</extensions>
15         </plugin>
16 </pluginManagement>
17 ...
18 <dependencyManagement>
19     <dependencies>
20         <dependency>
21             <groupId>org.mule.connectors</groupId>
22             <artifactId>mule-http-connector</artifactId>
23             <version>${http.connector.version}</version>
24             <classifier>mule-plugin</classifier>
25         </dependency>
26 </dependencyManagement>
27 ...
28 <distributionManagement>
29     <repository>
30         <id>anypoint-exchange-v3</id>
31         <name>Anypoint Exchange</name>
32         <url>https://maven......com/api/v3/org/${org}/maven</url>
33     </repository>
34 </distributionManagement>
36 <pluginRepositories>
37         <pluginRepository>
38             <id>mulesoft-releases</id>
39             <name>MuleSoft Releases Repository</name>
40             <layout>default</layout>
41             <url>https://repository.mulesoft.org/releases/</url>
42             <snapshots>
43                 <enabled>false</enabled>
44             </snapshots>
45         </pluginRepository>
```

1.11 Parent-POM

- Configure Parent-POM

1.12 Master-BOM

- Configure Master-BOM

1.13 Maven Build

- Use Anypoint Studio **7.12**
  - Update Workspace and Mule App
- Check and Change **local .m2** repository
- **Re-Install** Parent-POM & Master-BOM
- Build Mule Application using **Maven**
  - Understand the **401** Unauthorized Error

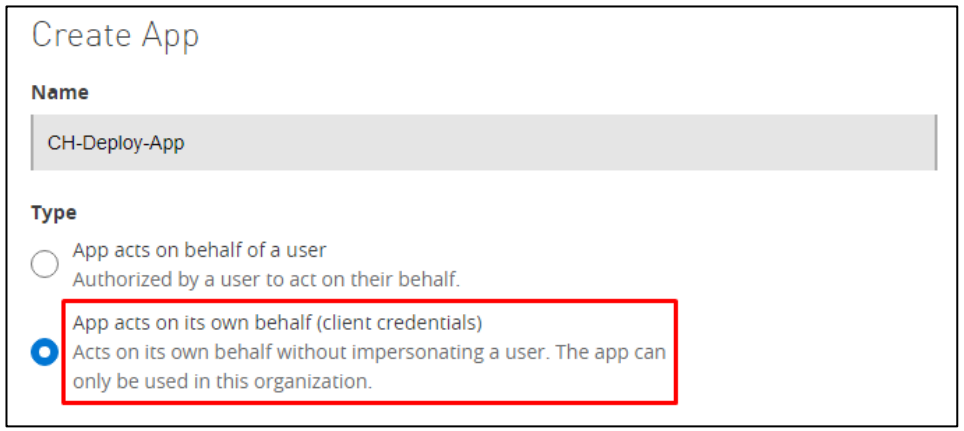
1.14 Connected App

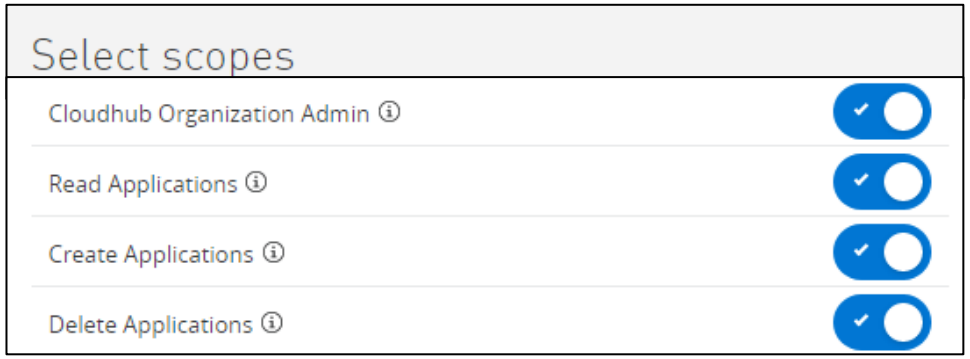
- What?
- Why?
- How?

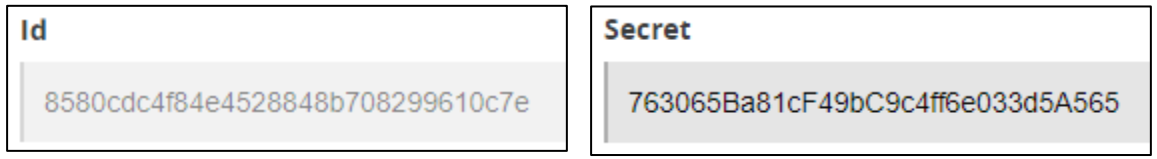
- Use credentials in settings.xml
- Build application

Mule Connected Apps

Connected Apps provides a framework that enables an external application to integrate with Anypoint Platform using APIs through **OAuth 2.0** and **OpenID** Connect







Scopes Required to **Publish and Consume** Artefacts from Exchange

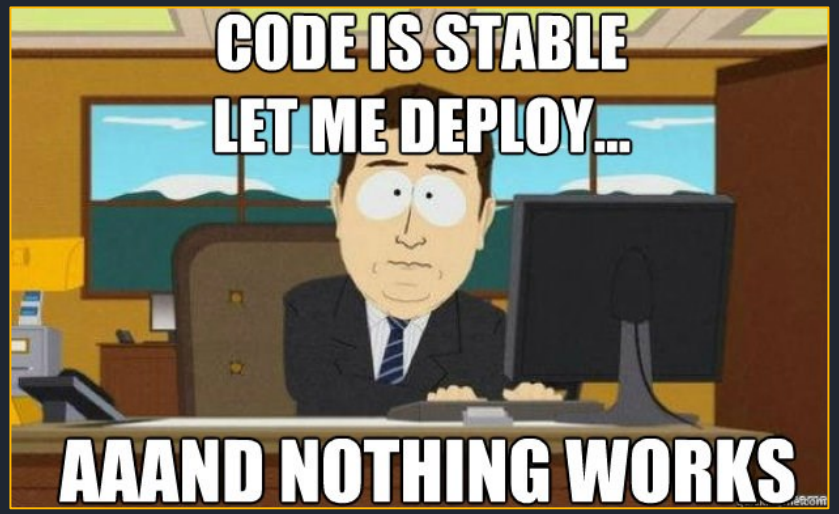
Scopes	Business Groups	
Exchange		
Exchange Contributor ⓘ	Super-Org-Mento	Manage
Exchange Viewer ⓘ	Super-Org-Mento	Manage
General		
Profile ⓘ		Remove

Connected Application Credentials in Maven **settings.xml**

```
<servers>
    <server>
        <id>Super-Org-Exchange-View-Write</id>
            <username>~~~Client~~~</username>
            <password>8a8f4a31a98d5405d633313e~?~26Bb0d83FD5e9a</password>
    </server>
</servers>
```

Maven Deployment

- CloudHub Deployment
- Connected App
  - Scopes



Maven CloudHub Deployment

```
mvn -DmuleDeploy deploy \
-Dap.client_id=65f06aff0bce38bb1932d \
-Dap.client_secret=Aa617859f30aA2e14eE2 \
-Dap.conapp.client_id=b92ce68b14c2766 \
-Dap.conapp.client_secret=16e58c1dD8ec71B \
-Ddeployment.env=dev \
-Dsecure.key=SecureKey007
```

muleDeploy

Instructs the plugin to deploy using the deployment strategy defined in the plugin configuration.

```
mvn mule:undeploy
```

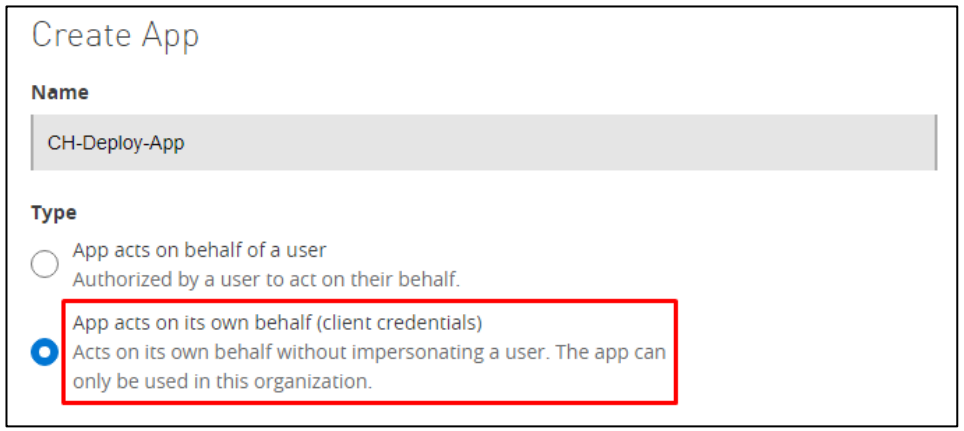
Authentication Modes	
User & Password	Username & Password
Server	settings.xml - Maven
Auth Token	authToken
Connected Apps	connectedAppClientId connectedAppClientSecret connectedAppGrantType

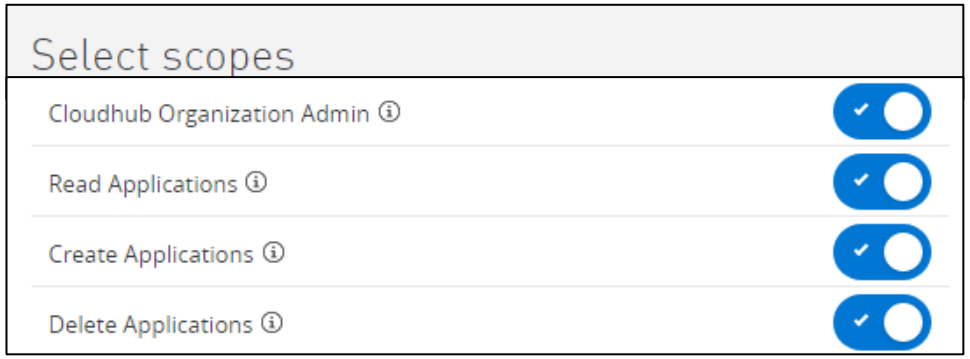
```xml
<properties>
  <deployment.prefix>sid-</deployment.prefix>
  <deployment.suffix>-${deployment.env}</deployment.suffix>
  <deployment.name>${deployment.prefix}${project.artifactId}${deployment.suffix}</deployment.name>
  <ap.environment>${deployment.env}</ap.environment>
```

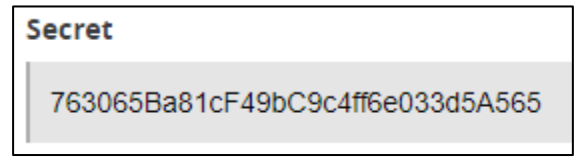
```xml
<plugin>
  <groupId>org.mule.tools.maven</groupId>
  <artifactId>mule-maven-plugin</artifactId>
  <!-- <version>3.5.3</version> -->
  <extensions>true</extensions>
  <configuration>
    <cloudHubDeployment>
      <businessGroup />
      <environment>${ap.environment}</environment>
      <region>us-east-2</region>
      <muleVersion>${app.runtime.semver}</muleVersion>
      <applyLatestRuntimePatch>true</applyLatestRuntimePatch>
      <workers>1</workers>
      <workerType>MICRO</workerType>
      <objectStoreV2>true</objectStoreV2>
      <applicationName>${deployment.name}</applicationName>
      <deploymentTimeout>600000</deploymentTimeout>
      <connectedAppClientId>${ap.conapp.client_id}</connectedAppClientId>
      <connectedAppClientSecret>${ap.conapp.client_secret}</connectedAppClientSecret>
      <connectedAppGrantType>client_credentials</connectedAppGrantType>
      <properties>
        <secure.key>${secure.key}</secure.key>
        <env>${deployment.env}</env>
        <anypoint.platform.client_id>${ap.client_id}</anypoint.platform.client_id>
        <anypoint.platform.client_secret>${ap.client_secret}</anypoint.platform.client_secret>
        <anypoint.platform.config.analytics.agent.enabled>true</anypoint.platform.config.analytics.agent.enabled>
        <anypoint.platform.visualizer.displayName>${project.artifactId}</anypoint.platform.visualizer.displayName>
        <anypoint.platform.visualizer.layer>${api.layer}</anypoint.platform.visualizer.layer>
      </properties>
    </cloudHubDeployment>
  </configuration>
</plugin>
```

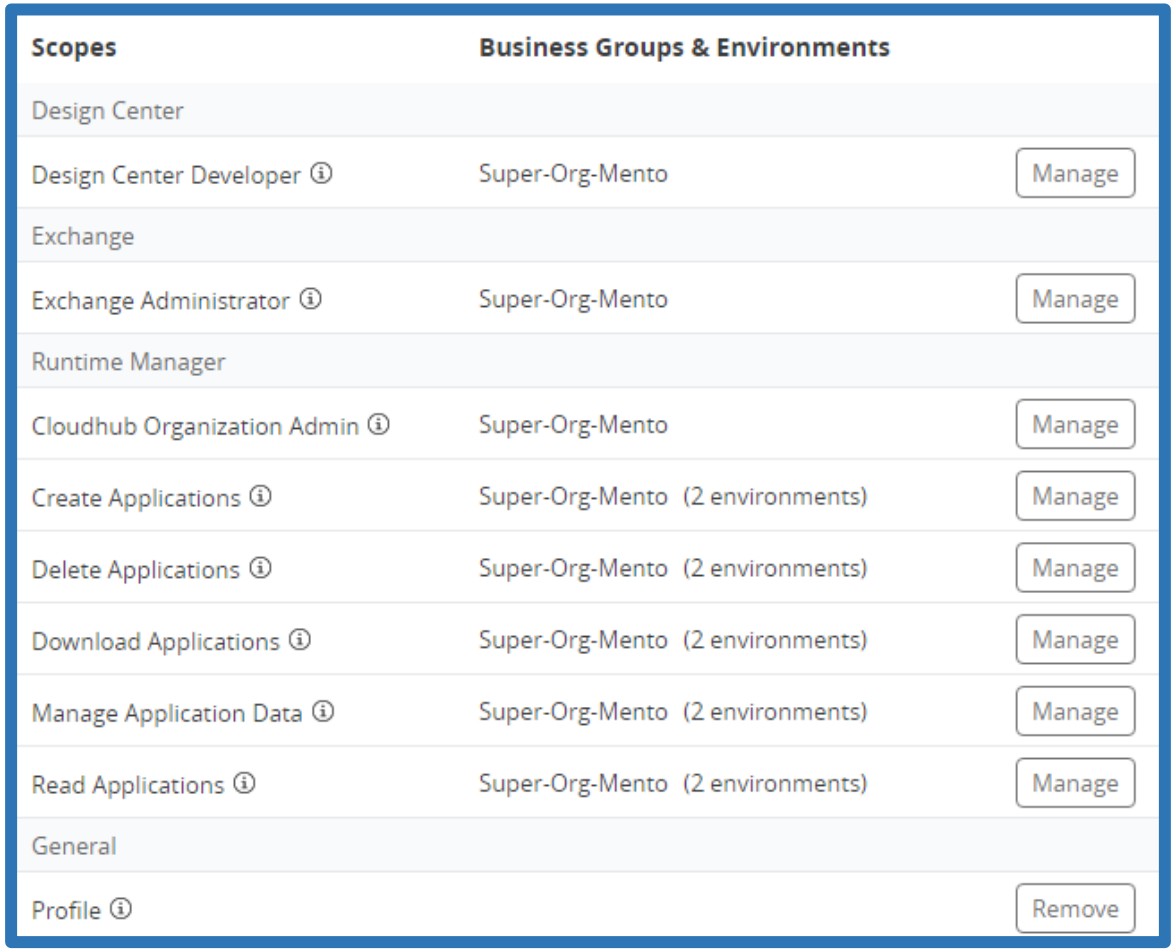
Mule Connected Apps

Connected Apps provides a framework that enables an external application to integrate with Anypoint Platform using APIs through **OAuth 2.0** and **OpenID** Connect



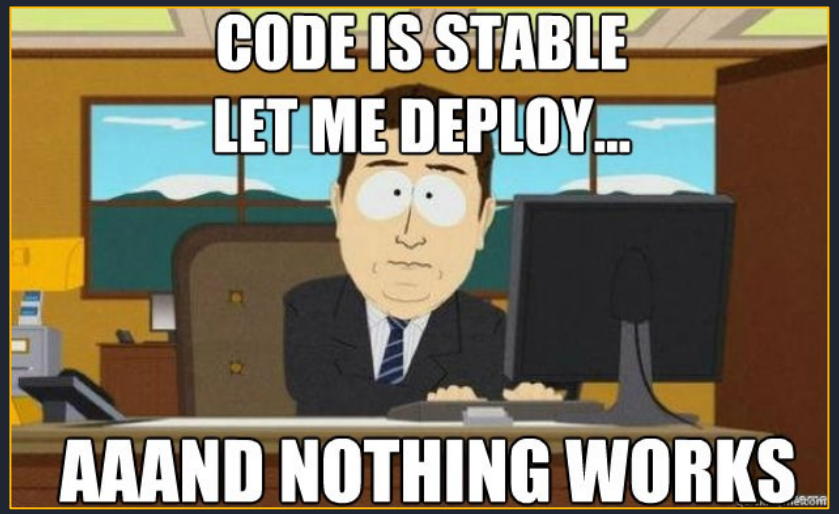






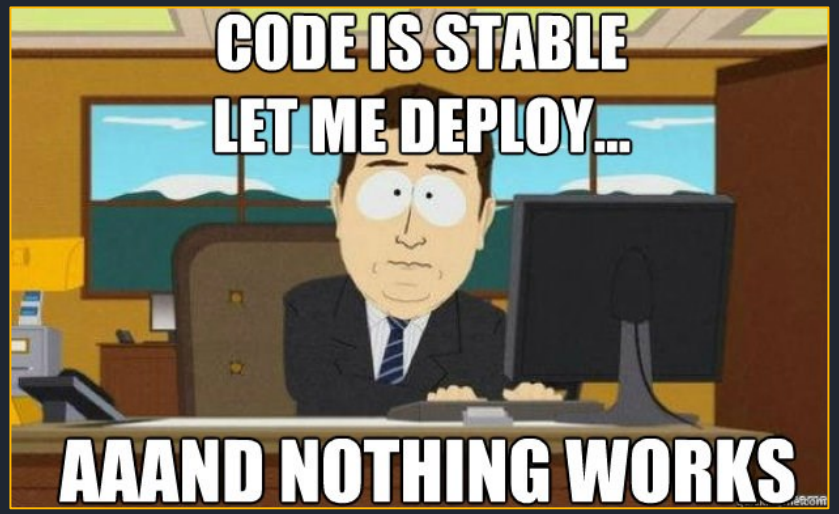
1.15 Maven Deployment

- CloudHub Deployment
- Connected App
  - Scopes



1.16 Anypoint Visulaizer

- What/Why
- Configure in application
- Deploy and check



Anypoint Visualizer

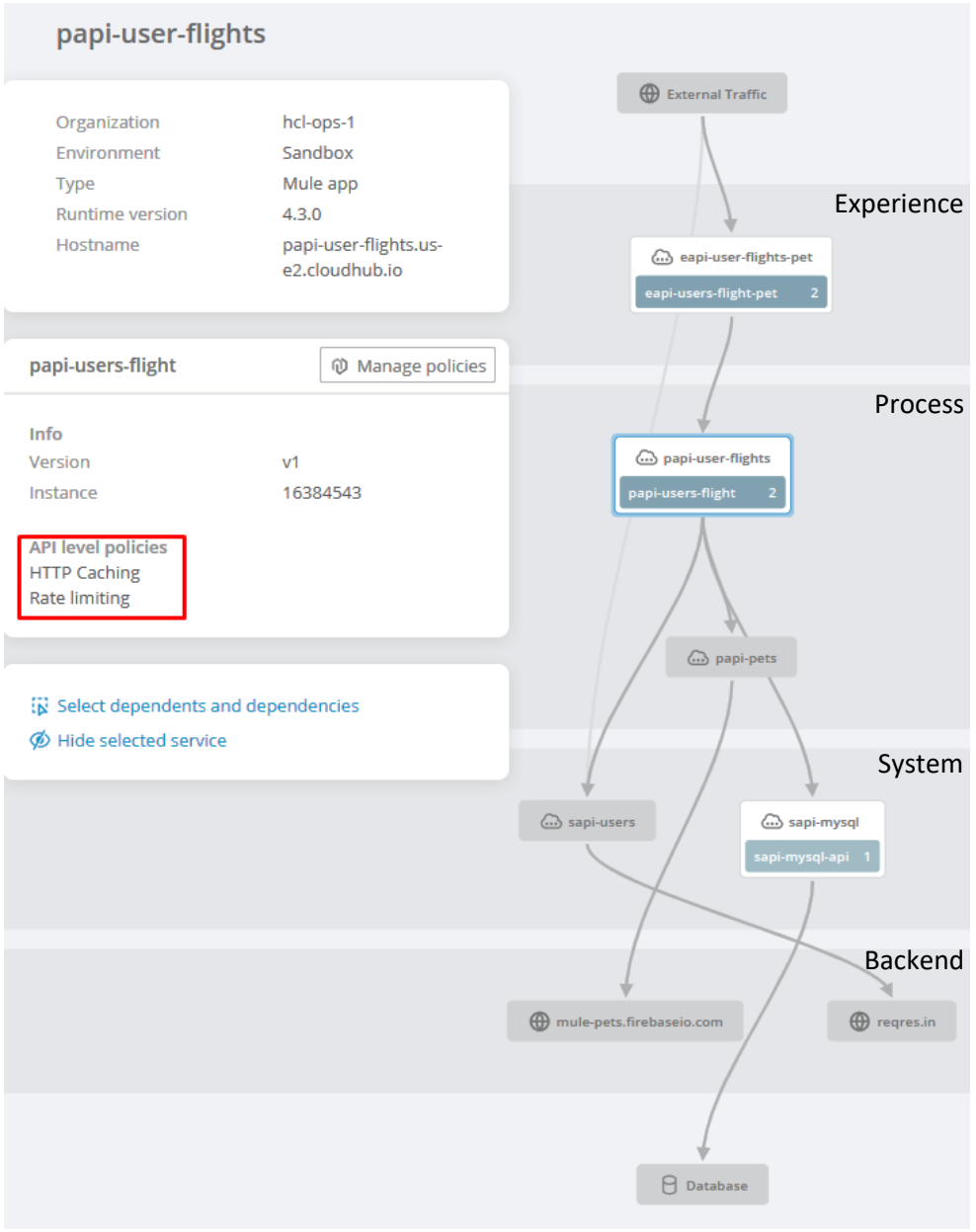
```
23        <api.visualizer.layer>Process</api.visualizer.layer>
24    </properties>
```

Mule app pom.xml

```
<anypoint.platform.config.analytics.agent.enabled>true</anypoint.platform.config.analytics.agent.enabled>
<anypoint.platform.visualizer.displayName>${project.artifactId}</anypoint.platform.visualizer.displayName>
<anypoint.platform.visualizer.layer>${api.visualizer.layer}</anypoint.platform.visualizer.layer>
</properties>
</cloudHubDeployment>
```

Parent-pom pom.xml

```
anypoint.platform.config.analytics.agent.enabled=true
anypoint.platform.config.analytics.agent.disabled=true
anypoint.platform.visualizer.displayName=<project-artifactId>
anypoint.platform.visualizer.layer=Process
```



Section 1 - Completed

#1 Setting Project Structure, Deployment Strategy and Development Best Practises

- Add OAS to Design Center and Publish to Exchange
- Configure API Instance in API Manager
- Import API from Exchange to Anypoint Studio
- Configure API Autodiscovery
- Coding Conventions
- Secure Communication – Cryptography
- Self-signed Certification creation
- CloudHub Architecture
- Configuration Properties
- Secure Properties
- Hidden Properties
- Maven Resource Filtering
- Reducing Build Redundancy
- Maven Basic
- CloudHub Maven Deployment
- Anypoint Visualizer

MuleSoft Certified Developer - Level 2 (Mule 4)

MCD2

Expose production-ready Anypoint Platform-managed APIs from Mule applications

- Implement versioning of specific API-related artifacts
- Configure custom or out-of-the-box (OOTB) API policies
- Implement server-side caching of API invocations using API policies
- Request access to APIs while honoring environment-specific scoping
- Implement HTTP callbacks (webhooks)
- Code API implementations to perform API auto-discovery

Implement performant and reliable Mule applications

- Implement ObjectStore persistence for all Mule deployment options
- Implement fault-tolerant, performant, and traceable message passing with the VM and AnypointMQ connectors
- Implement fault-tolerant invocations of HTTP-based APIs, reacting correctly to HTTP status codes
- Validate assertions using the Validation module
- Validate messages against XML- or JSON-Schema documents
- Parallelize integration logic using scatter-gather
- Implement compensating transactions for partially failed scatter-gather
- Implement client-side caching of API invocations for performance

Implement monitorable Mule applications

- Expose healthcheck endpoints from a Mule application
- Implement effective logging
- Change log levels and aggregate and analyze logs
- Monitor Mule applications and Mule runtimes using Anypoint Platform or external tools
- Implement message correlation, including persistence and propagation of correlation IDs over HTTP

Implement maintainable and modular Mule applications and their Maven builds

- Modularize and optimize Mule application Maven build configurations
- Implement Maven-based automated deployment to Mule runtimes
- Execute MUnit tests with Maven
- Implement unit tests with the MUnit framework and tooling
- Build custom API policies
- Encapsulate common Mule application functionality in libraries
- Implement custom message processors using the Mule SDK or XML SDK

Secure data at rest and in transit

- Implement secure, environment-dependent properties management
- Create, package, and distribute keys and certificates
- Expose and invoke APIs over HTTPS
- Implement TLS mutual authentication on the client and server side
- Implement API invocations authenticated by Basic Auth or OAuth2 with HTTP or REST connectors

MuleSoft Certified Developer - Level 2 (Mule 4)

2 hours Virtual

Summary

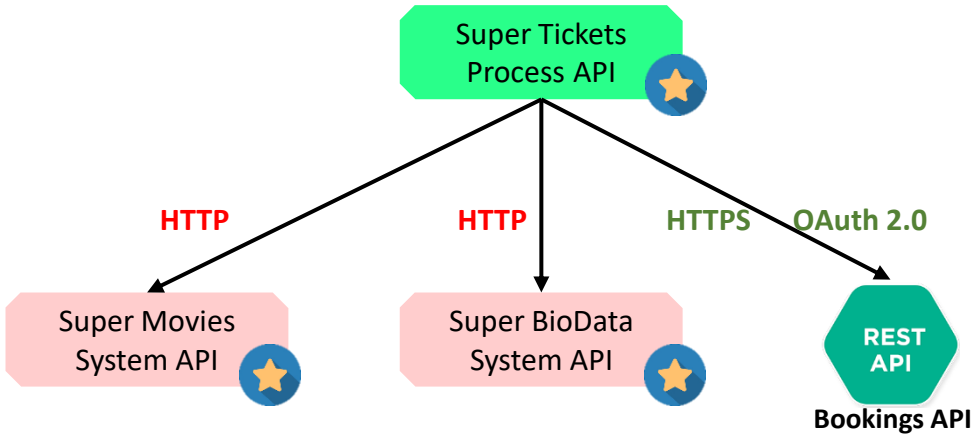
COMING SOON! Come back soon to register for this new certification exam!

A *MuleSoft Certified Developer – Level 2* should be able to independently work on production-ready Mule applications – applications that are ready to be used in a DevOps environment in professional software development projects and address and balance critical non-functional requirements including monitoring, performance, maintainability, reliability, and security. The *MuleSoft Certified Developer – Level 2* exam validates that a developer has the required knowledge and skills to:

- Expose production-ready Anypoint Platform-managed APIs from Mule applications
- Implement maintainable and modular Mule applications and their Maven builds
- Implement monitorable Mule applications
- Implement performant and reliable Mule applications
- Secure data at rest and in transit

#2 Mule Integration - API Development, Error Handling, Caching Strategies

- Add System API REST Connectors to Anypoint Studio
- Get credentials and invoke the SAPI
- Configure HTTP Non-functional requirement
- OAuth 2.0 – Client Credentials Grant Type
- Add DEBUG Async Logging
- Add transformation logic
- Handle custom exceptions
- Parallel execution using Scatter-gather
- Scatter-gather error handling
- Use Until Successful scope to handle HTTP Errors
- Transient vs Permanent HTTP Errors
- Client Side Caching using Object Store/Cache Scope
- Server Side Caching using API Manager's API Caching policy

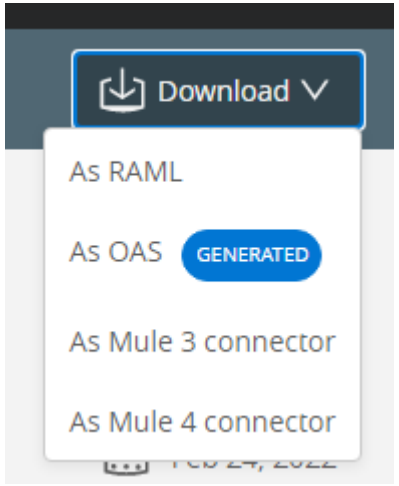


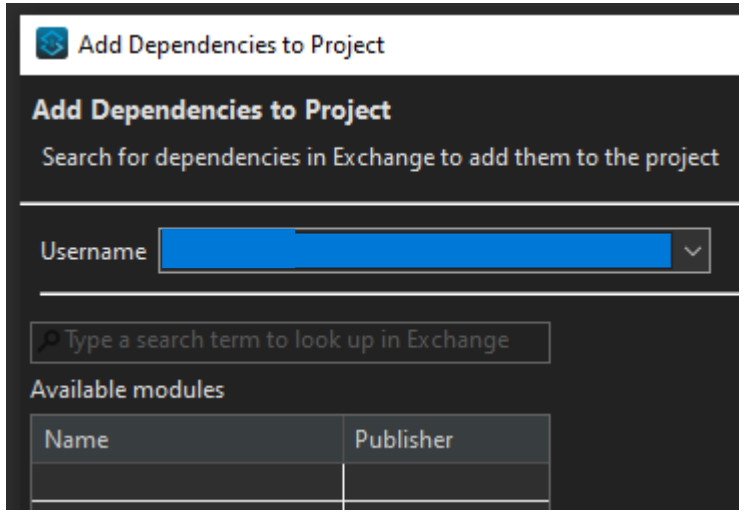
REST Connect

- What?
- How to create?
- How to configure in Mule app?

REST Connect

REST Connect DOES NOT currently support custom TLS configurations





Supported Formats

- OAS 3 - JSON
- OAS 3 - YAML
- OAS 2 - JSON
- OAS 2 - YAML
- RAML 1.0

Supported Security Schemas

- Basic Authentication
- OAuth 2.0 Client Credentials
- OAuth 2.0 Authorization Code
- Pass Through
- Digest Authentication

Changing Connector Name

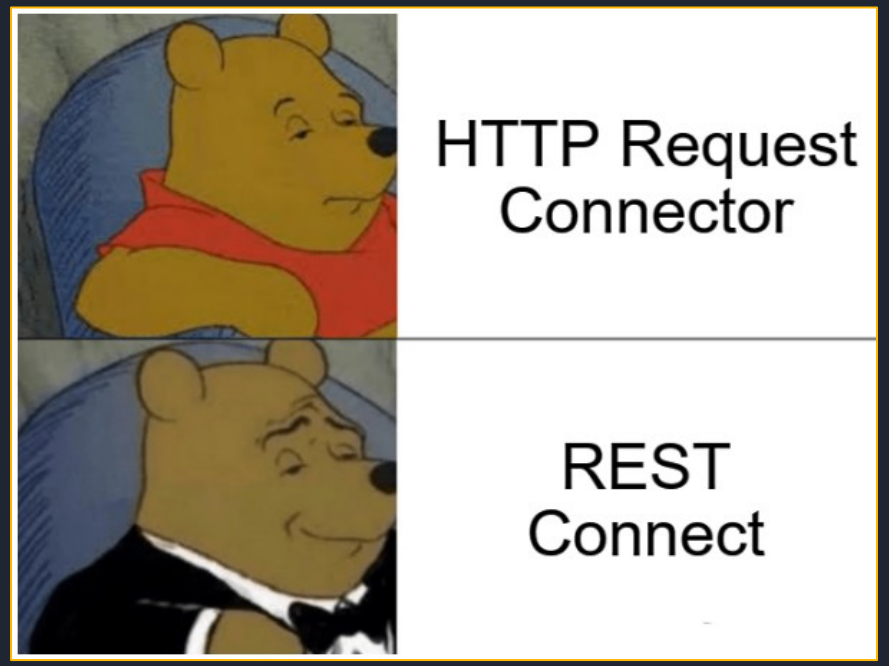
- displayName
- REST Connect library

```
uses:
  rest-connect: exchange_modules/org.mule.connectivity/rest-connect-library/1.0.0/rest-connect-library.raml

get:
  (rest-connect.operationName): Retrieve a comment by id
  responses:
    200:
      body:
       type: Comment
```

Example REST Connect library

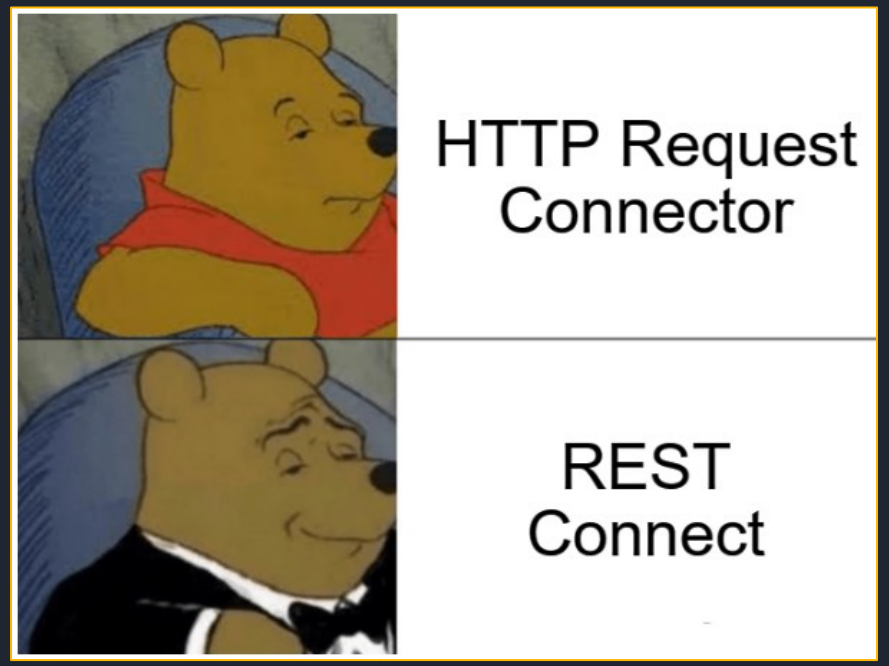
2.1 REST Connect

- How to configure in Mule app?



2.2 REST Connect

- Get Access (client id and secret)
- Configure them in App

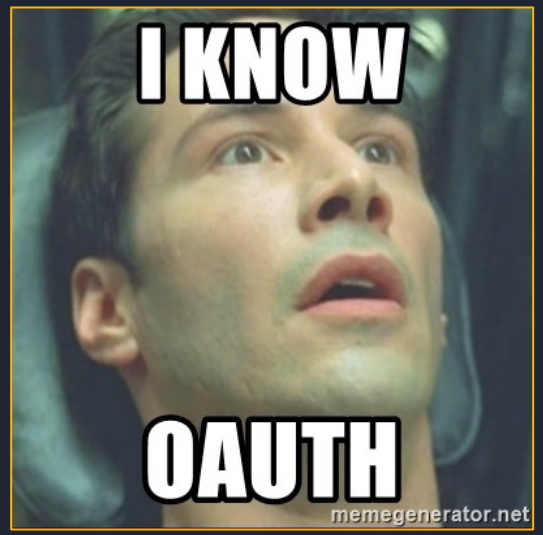


2.3 REST Connect

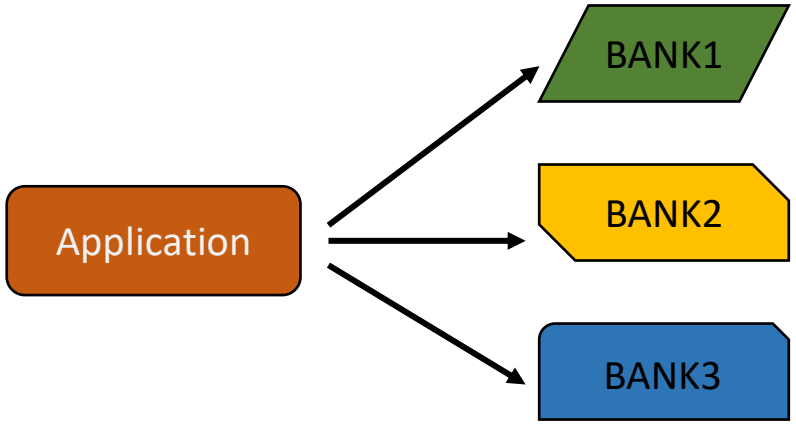
- Add timeouts to follow
  - Non-functional requirement

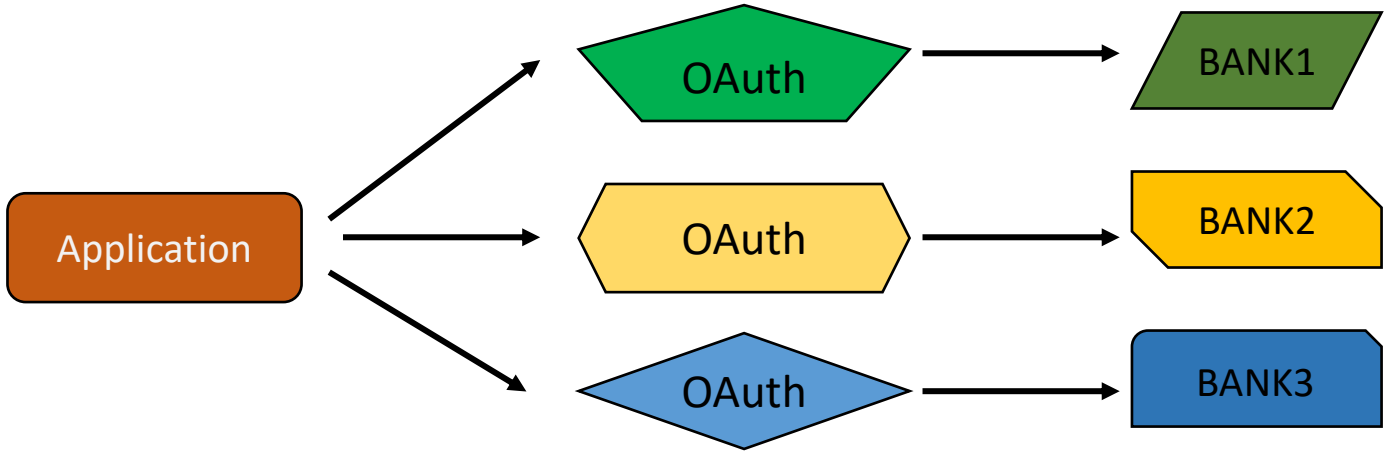
OAuth 2.0

- What/Why/How?
- Authorization Code Grant type steps



OAuth – Open Authorization





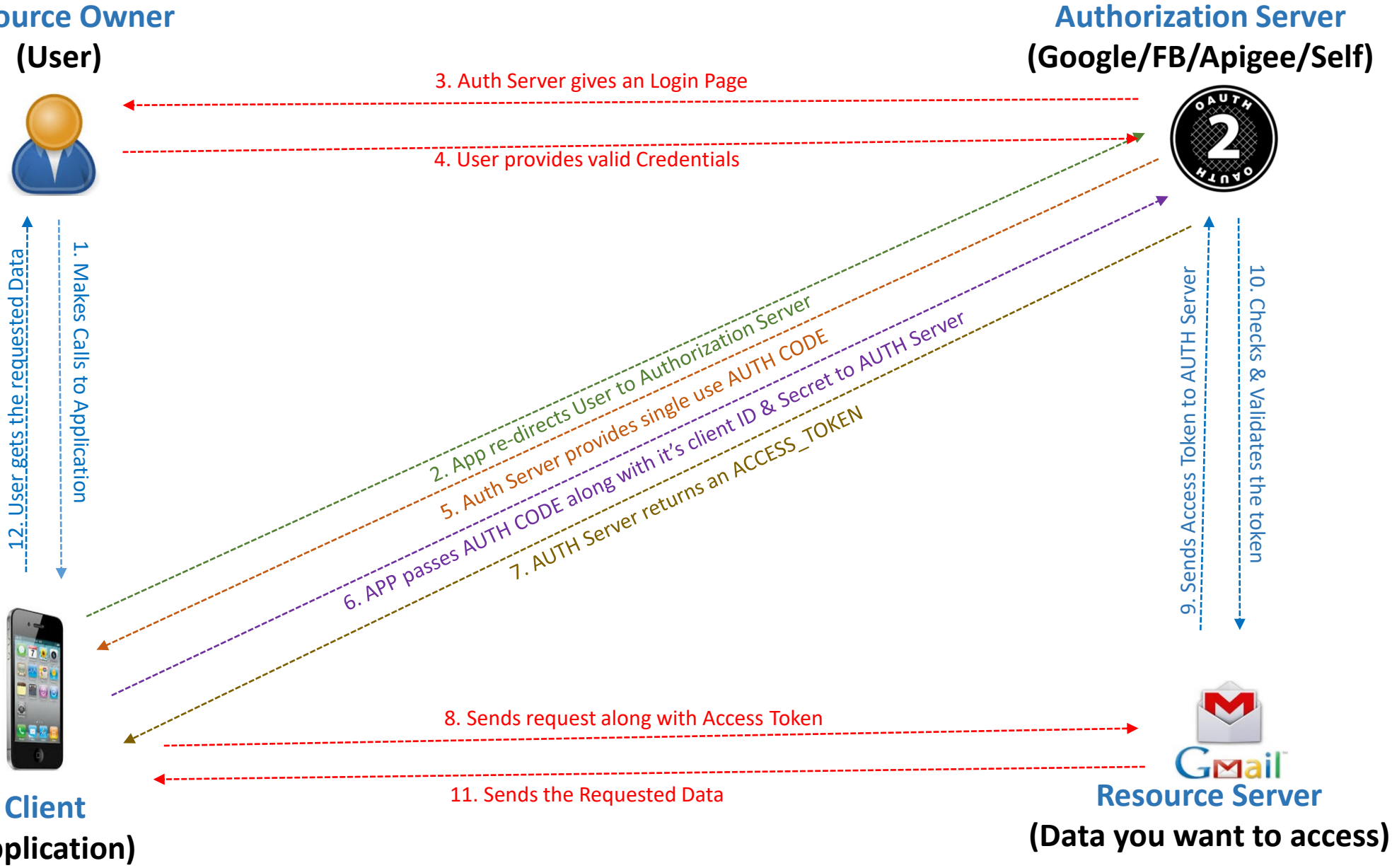
Authentication = Hotel Reception

Authorization = Key to Room

- Access Token — Refresh Token — Opaque Tokens
- Scope — read — write — admin — user
- Applications — Client ID — Client Secret

```
curl -H "Authorization: Bearer UAj2yiGAcMZGxfN2DhcUbl9v8WsR" \
     http://api.sid.com/v1/weather/forecast?w=12797282
```

OAuth – Authorization Code Grant Type Flow

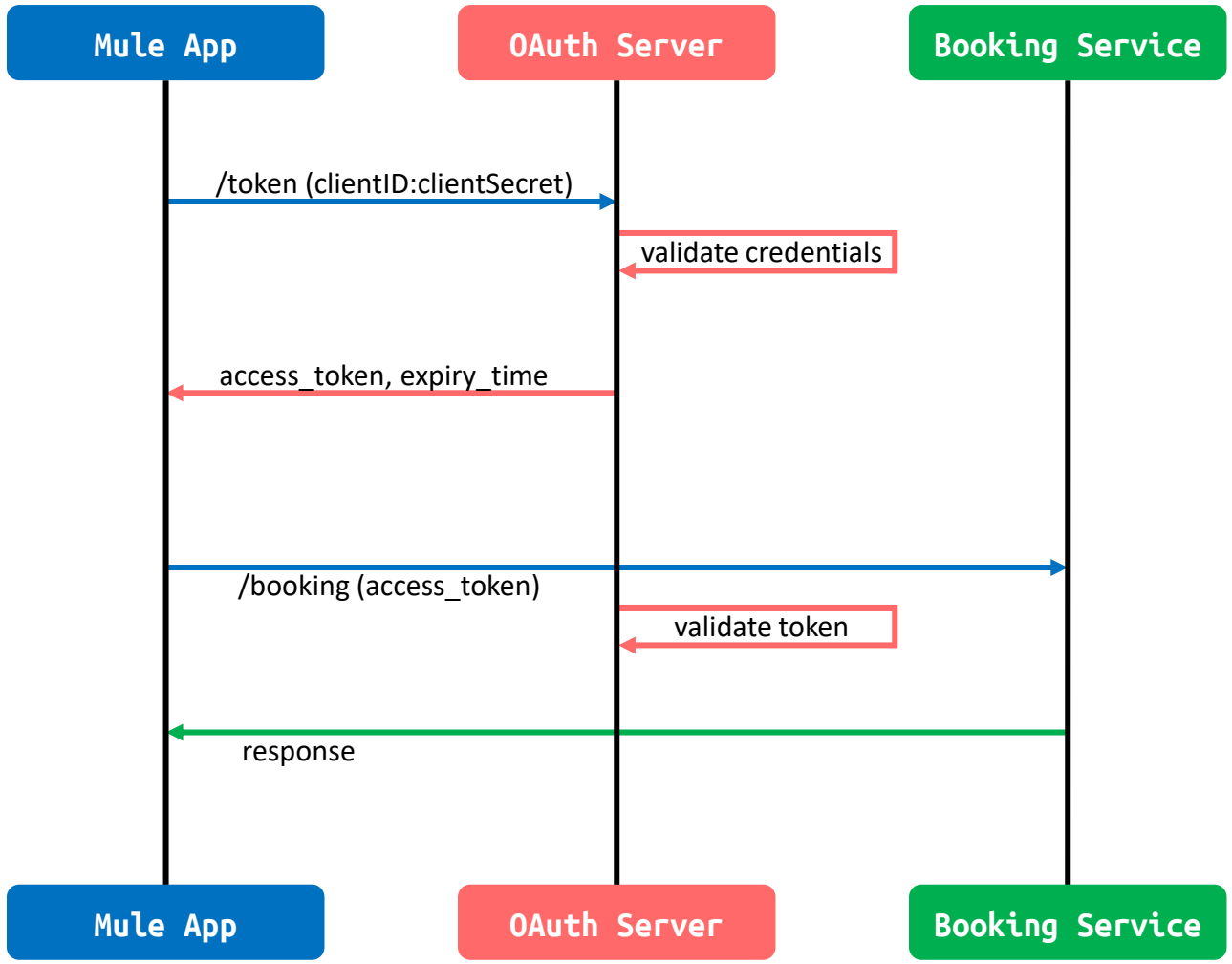


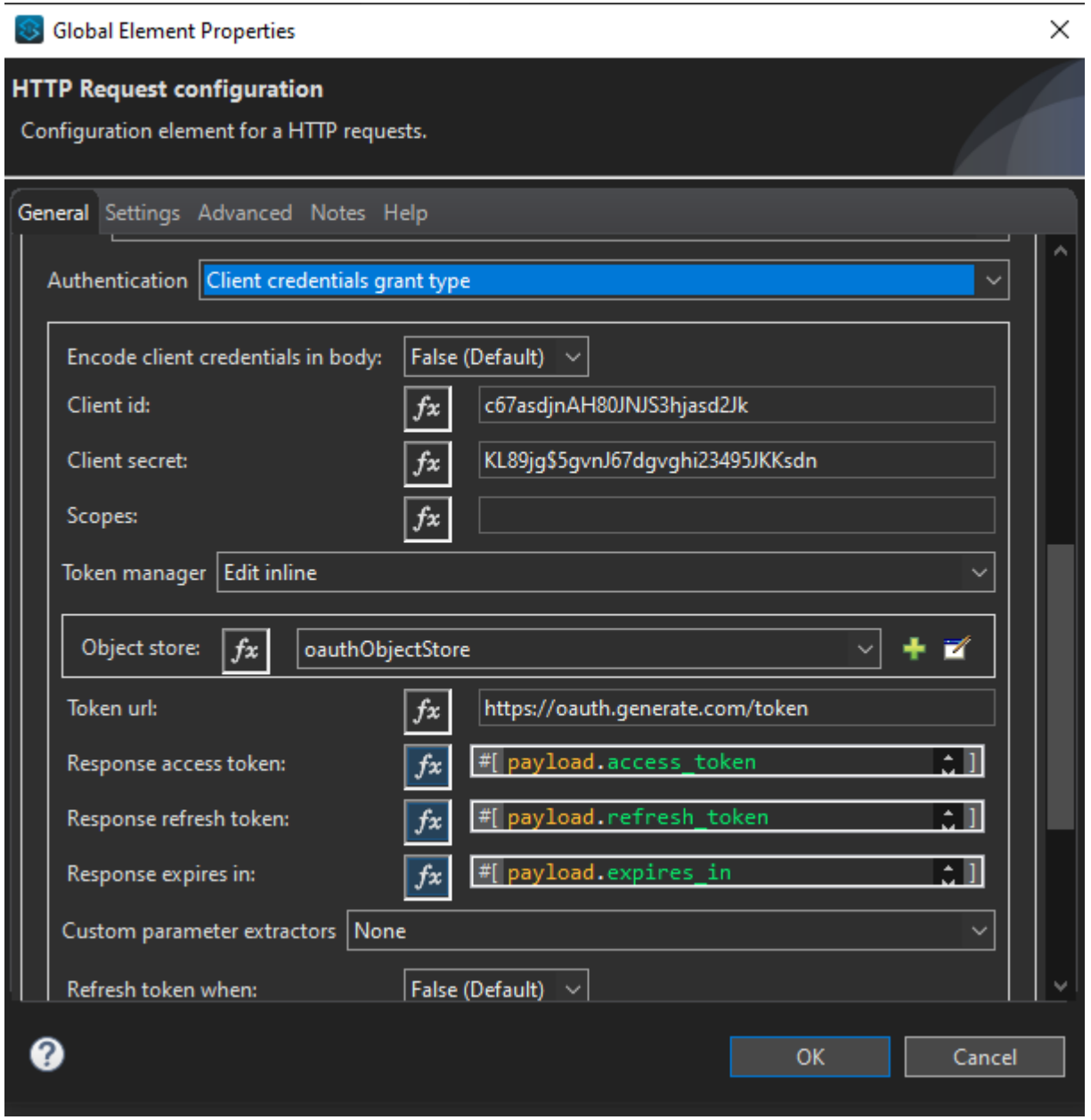
OAuth 2.0

- Configure OAuth in HTTP Request Connector

OAuth Config in Mule HTTP Request Connector

Client Credentials Grant Type



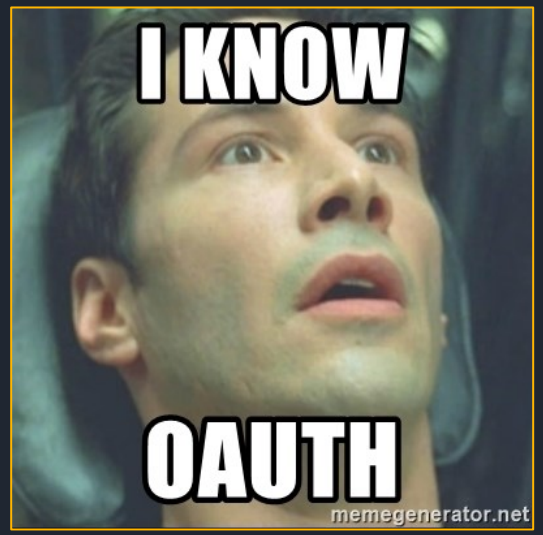


2.4 OAuth 2.0

- Understand how Oauth token is retrieved
- Configure OAuth in Mule Application

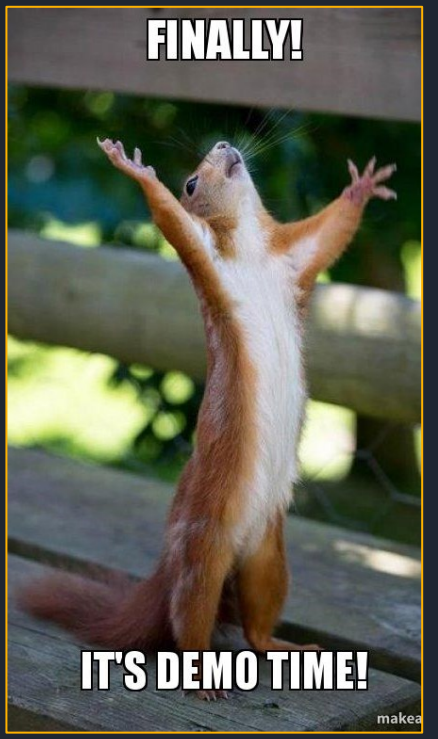
2.5 OAuth 2.0

- Check Mule App OAuth interaction in LOGs
  - Change log level - DEBUG



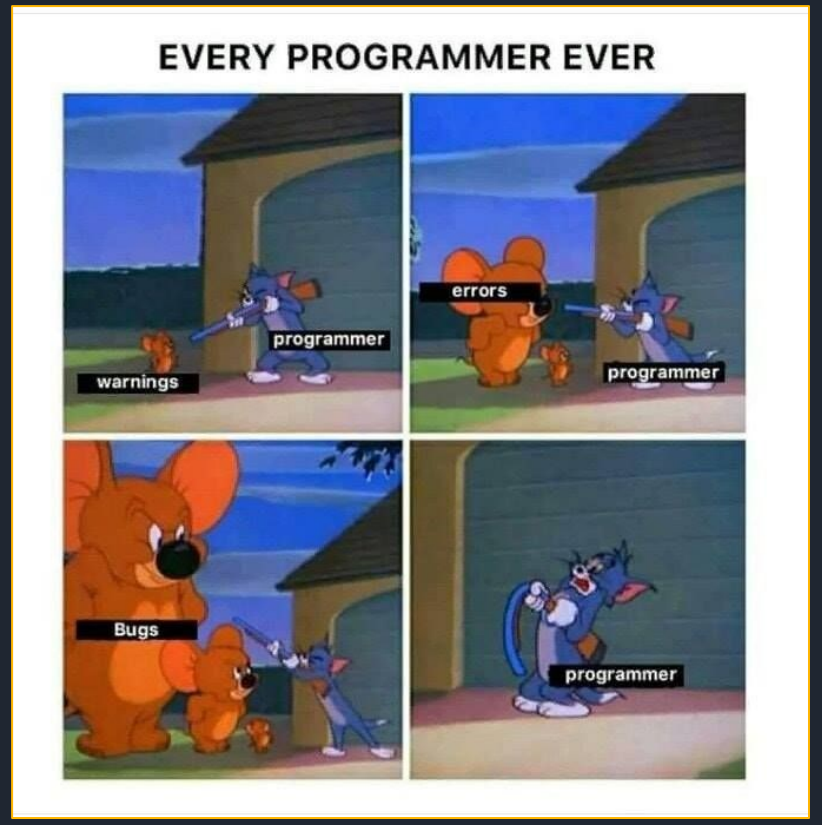
2.6 Response Modification

- Add transformation logic



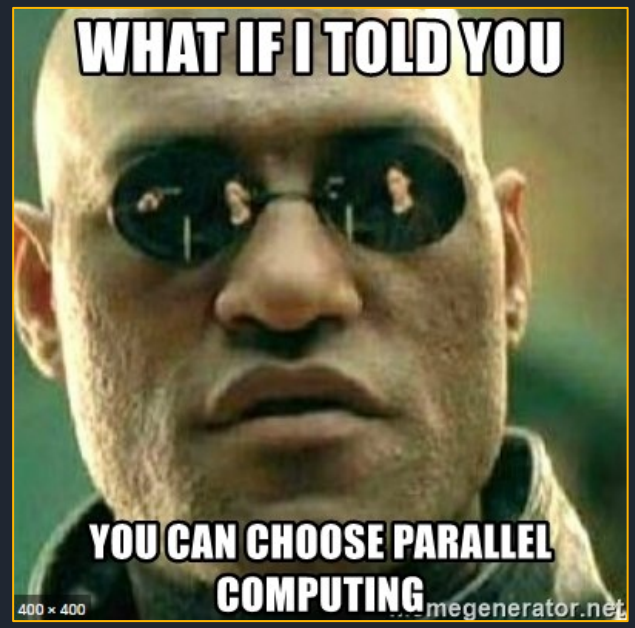
2.7 Error Handling

- Handling custom errors

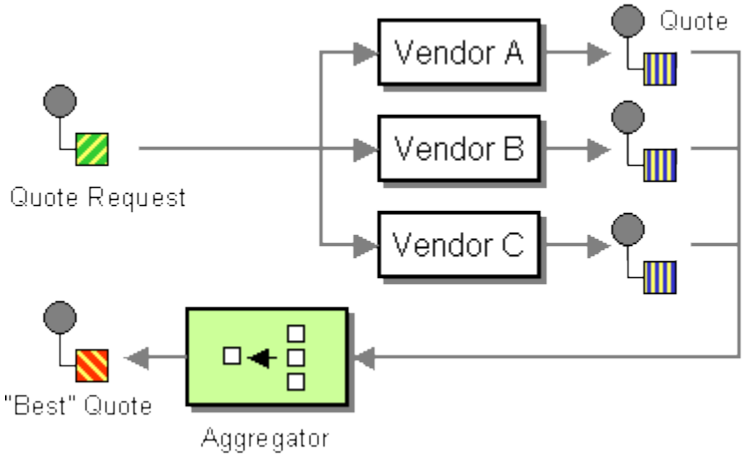


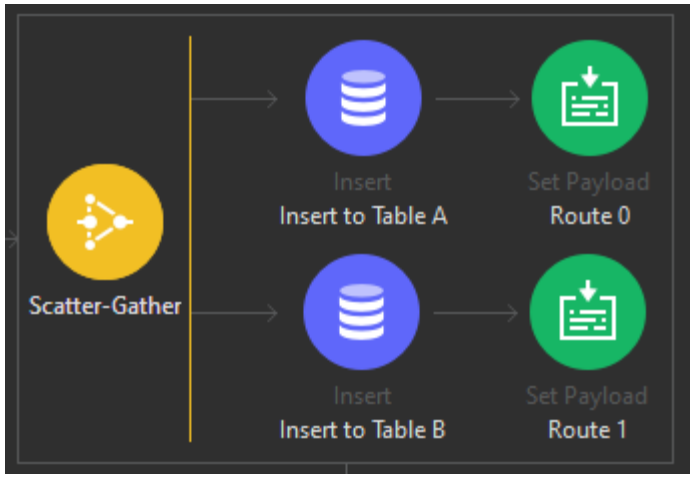
Parallel Processing

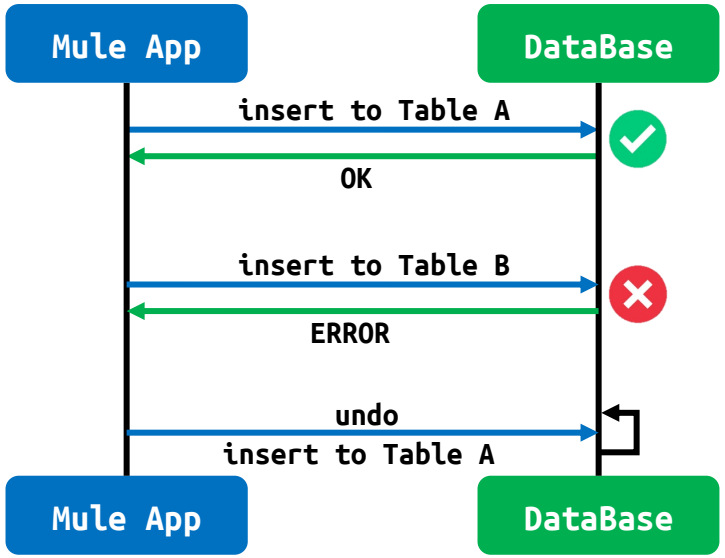
- Use Scatter-Gather
- Error Handling



Scatter Gather







if maxConcurrency = 1, then Scatter-gather processes the routes **sequentially**

if a **route times out**, raises a **MULE:TIMEOUT**

if a **route fails**, raises a **MULE:COMPOSITE_ERROR**

When running within a **transaction**, Scatter Gather **does not execute in parallel**

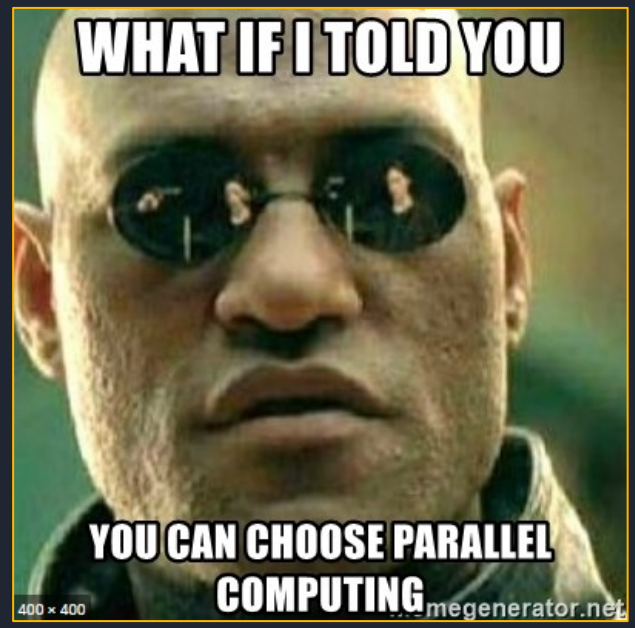
if Insert to Table B fails, Compensation Logic for Table A

`#[error.errorMessage.payload.failures]` → Route 1

`#[error.errorMessage.payload.results]` → Route 0

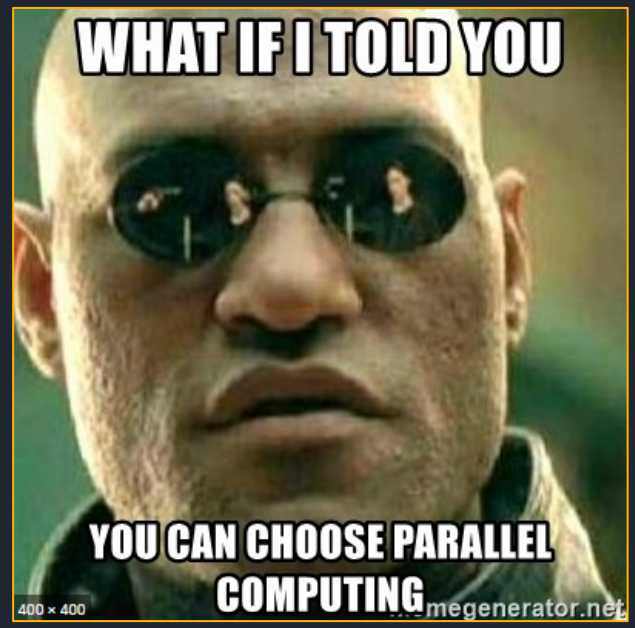
2.8 Parallel Processing

- Use Scatter-Gather



2.9 Parallel Processing

- Use Scatter-Gather
  - Get Failed Routes
  - Get Success Routes



2.10 Parallel Processing

- Use Scatter-Gather
  - Compensation Logic

Errors and Retry

- Using Until Successful Router
- Handle HTTP Errors
  - Transient Errors
  - Permanent Errors

Errors and Retry

2xx Success	
200	Success
201	Created
204	No-Content

3xx Redirection	
301	Permanent Redirect
302	Temporary Redirect
304	Not Modified

4xx Client Error	
401	Unauthorized Error
403	Forbidden
405	Method Not Allowed
429	Too Many Requests

5xx Server Error	
501	Not Implemented
502	Bad Gateway
503	Service Unavailable
504	Gateway Timeout

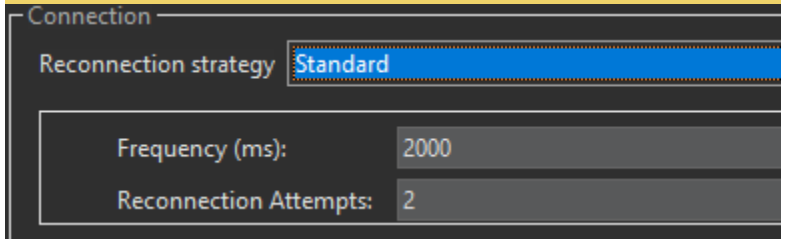
Transient Errors

429 | 503 | 504

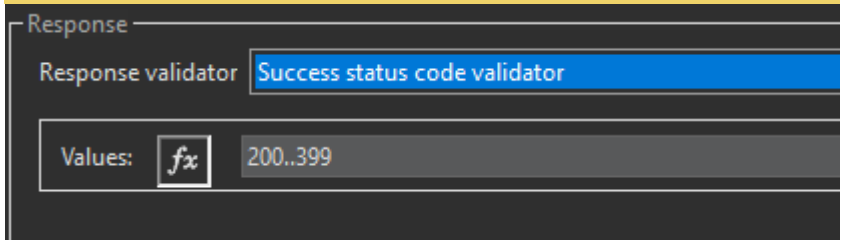
Permanent Errors

401 | 403 | 405 | 501

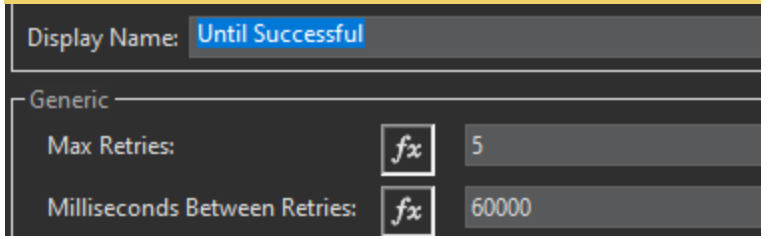
Reconnection Strategy



HTTP Request Connector



Until Successful Scope

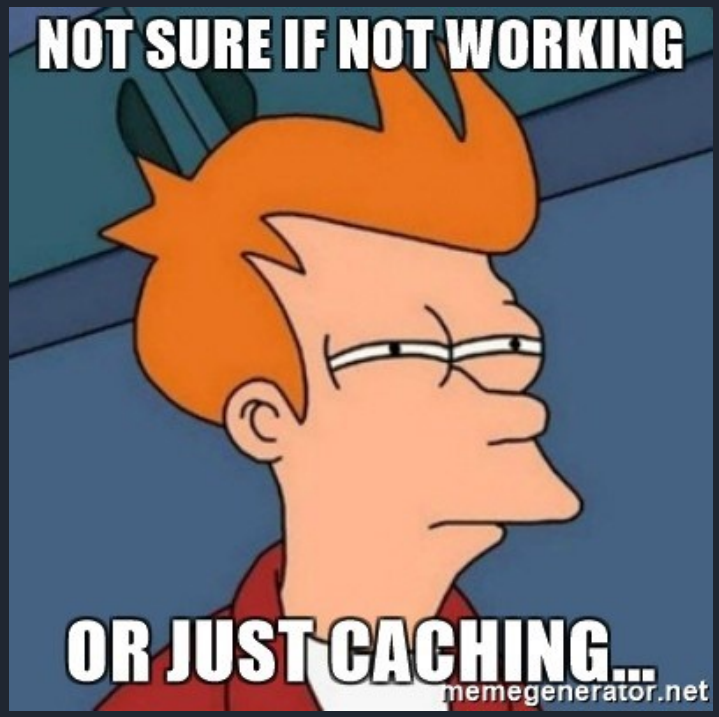


2.11 Errors and Retry

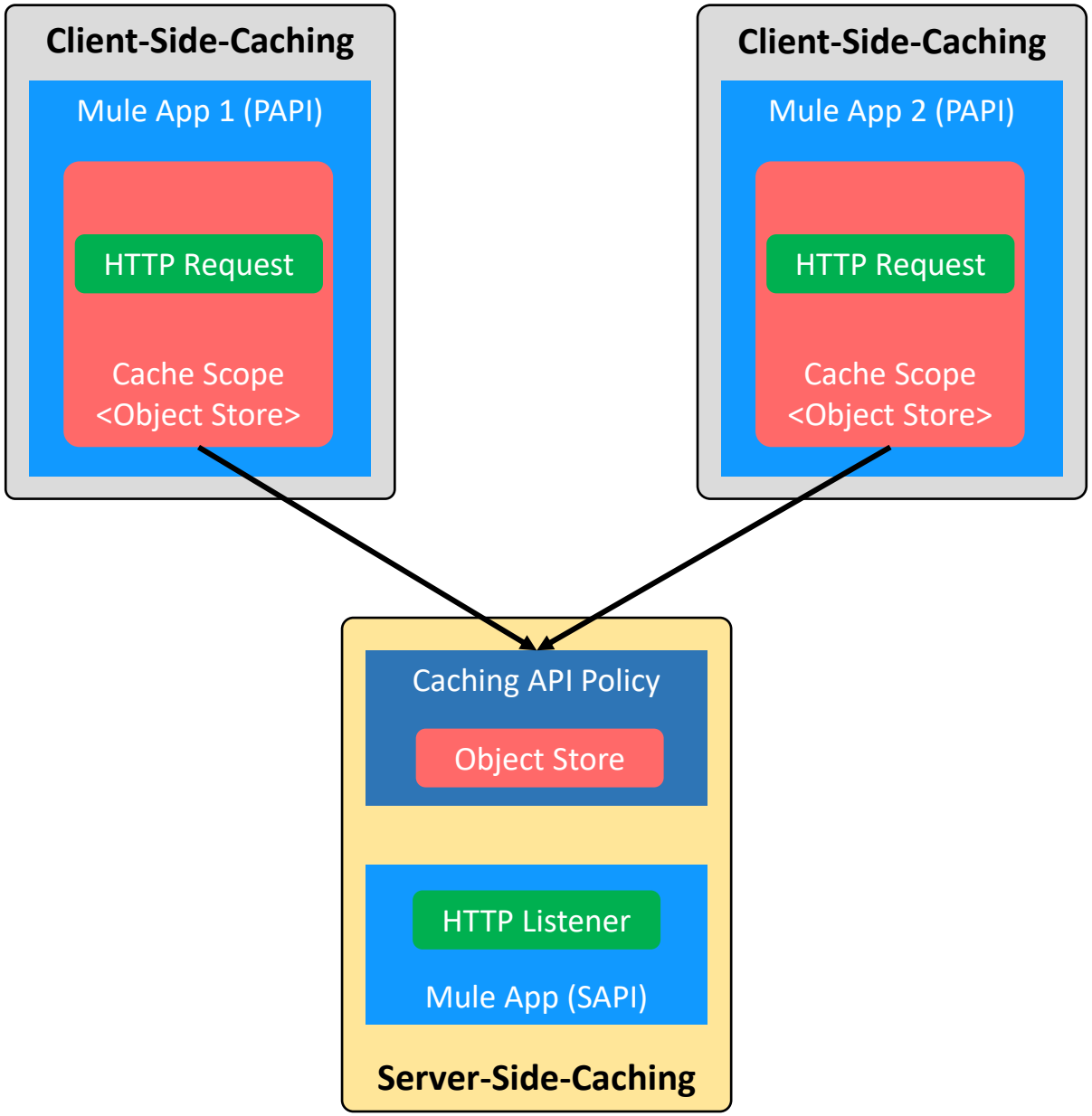
- Using Until Successful Router
- Handle HTTP Errors
  - Transient Errors
  - Permanent Errors

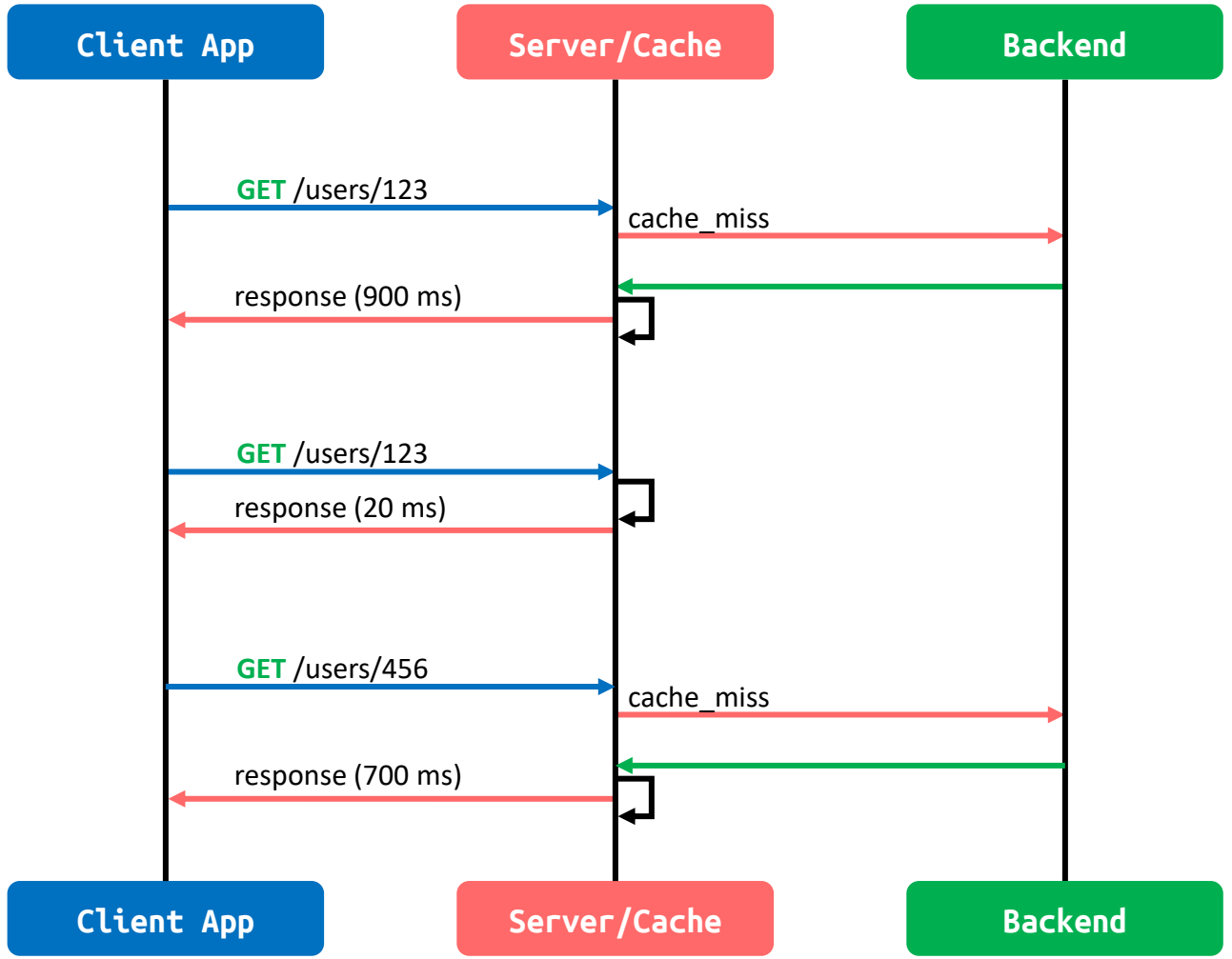
Caching

- What is caching?
- Type of caching
  - Server-side
  - Client-side
- Cache Scope
- Object Store



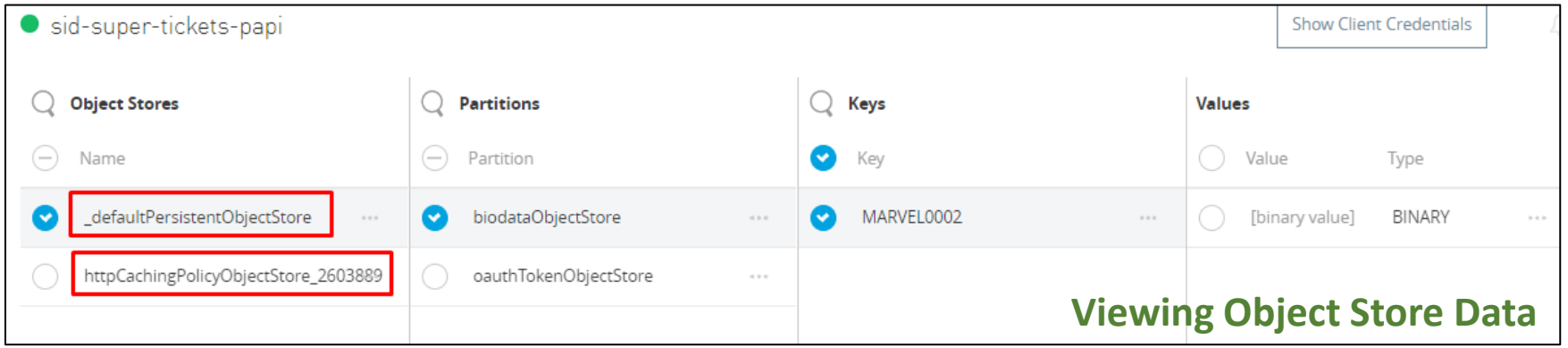
Caching


Client-Side-Caching
Mule App 1 (PAPI)
HTTP Request
Cache Scope
<Object Store>
Client-Side-Caching
Mule App 2 (PAPI)
HTTP Request
Cache Scope
<Object Store>
Caching API Policy
Object Store
HTTP Listener
Mule App (SAPI)
Server-Side-Caching

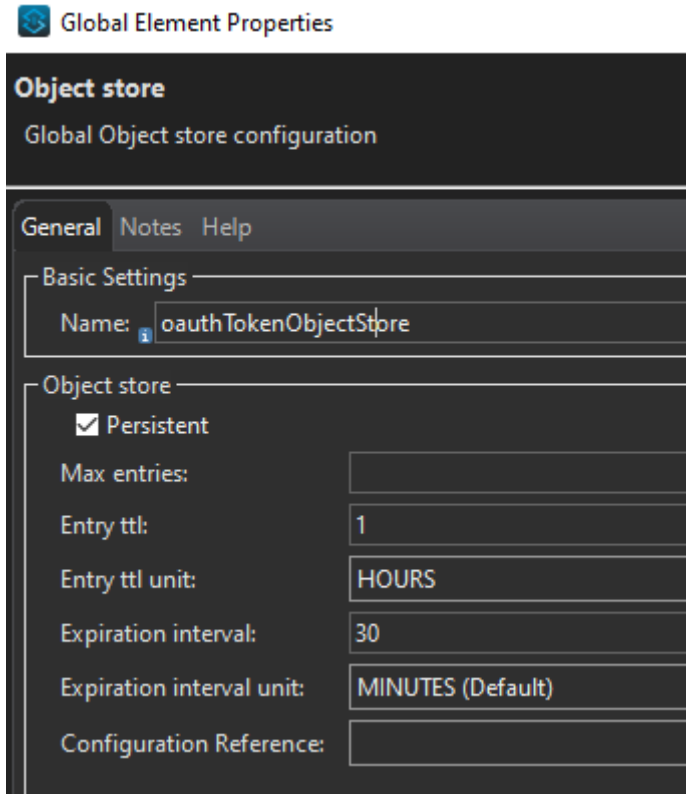

Client App
Server/Cache
Backend
GET /users/123
cache_miss
response (900 ms)
GET /users/123
response (20 ms)
GET /users/456
cache_miss
response (700 ms)
Client App
Server/Cache
Backend

Object Store

	Anypoint Object Store V2	CloudHub Object Store V1	Mule Object Store
number of entries	Unlimited	100,000 per application	no limit
number of key:value pairs			no limit
key size	n.a	768 byte key size	no limit
value size	**10 MB**	**1 MB**	**no limit**
data per application	Unlimited	1 GB	no limit
Supported by	Object Store Connector | Mule 4	Object Store Connector | Mule 3 & 4	Object Store Connector
API Rate Limit	Standard - 10TPS | Premium - 100TPS	No API Rate limit	n.a



Viewing Object Store Data



If **entryTTL** is not set, **TTL** is rolling and as long as the object is accessed at least once a week, the TTL will extend for another **30 days**

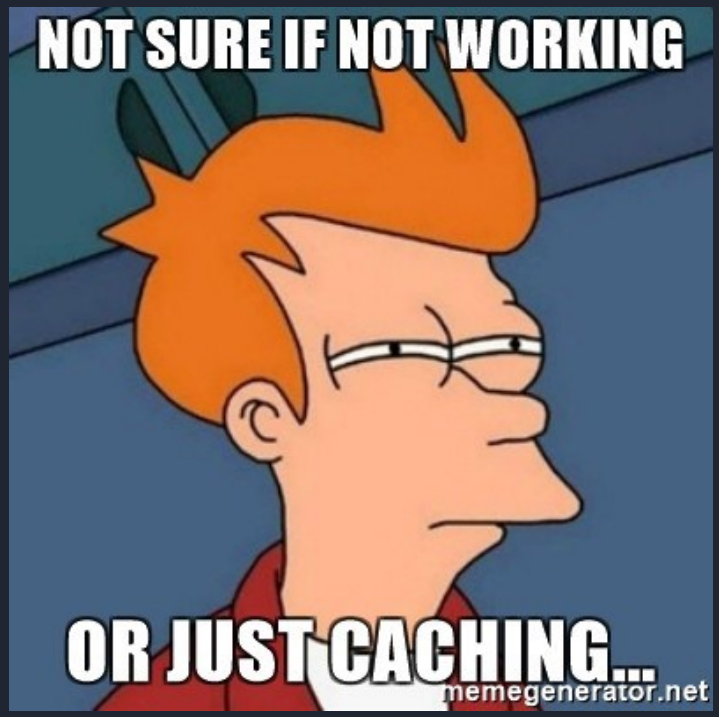
An **expiration thread** runs every 30 minutes and discards the elements that exceed their **time to live (TTL)** or their maxEntries limit.

OSv2 can be accessed using **Object Store connector** and/or **OSv2 REST API**

A non-persistent object store does not use the **OSv2** service

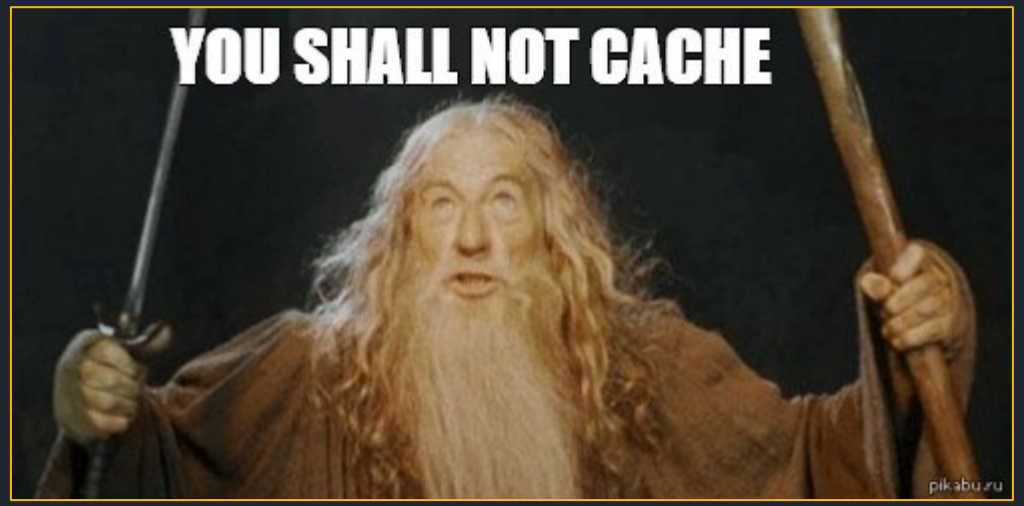
2.12 Client-side Cache

- Cache Scope
- Object Store

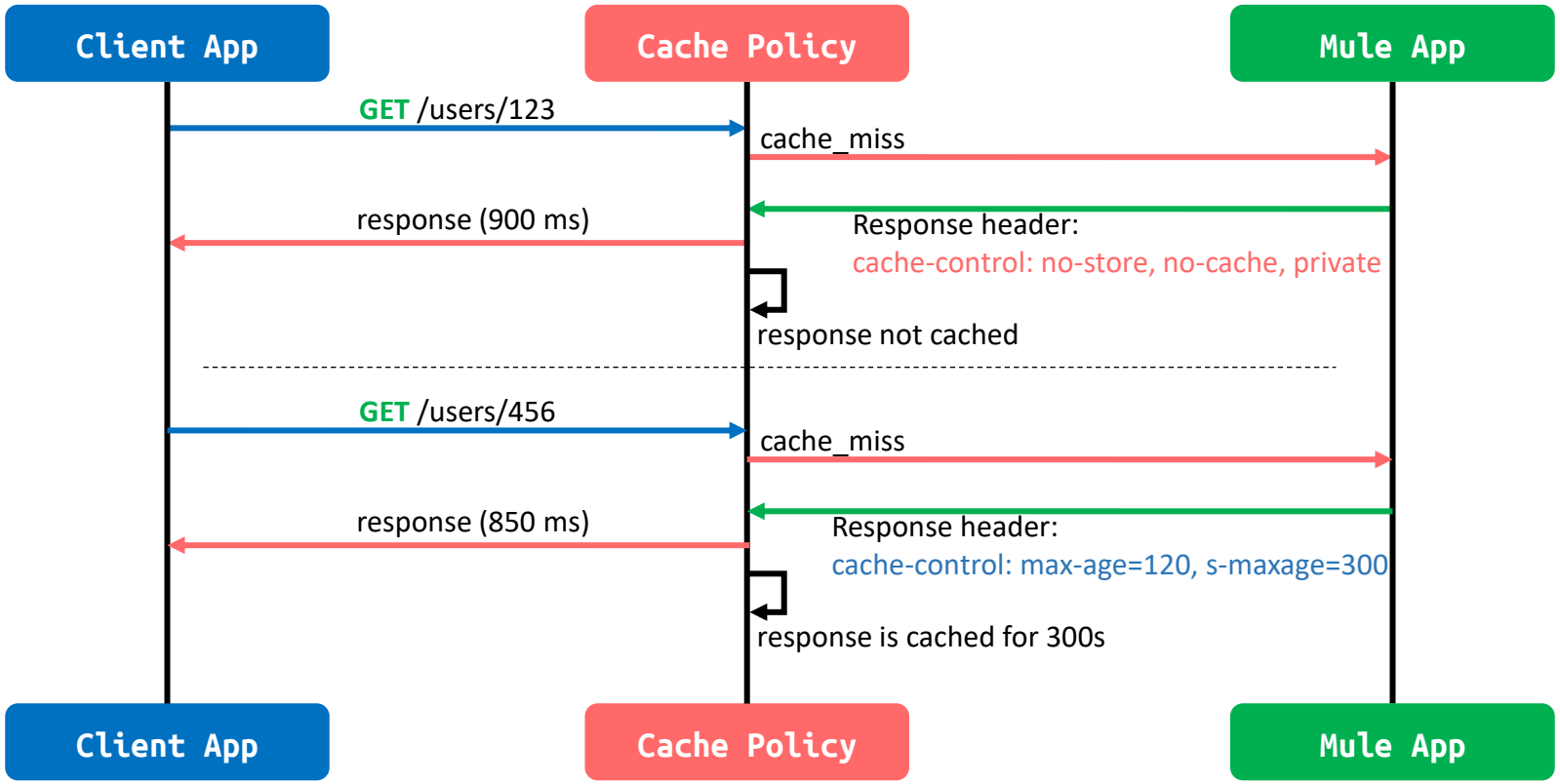


Sever-Side Caching

- API Caching Policy
- Invalidate Header



API Caching Policy



Configuration Detail

Policy	HTTP Caching
HTTP Caching Key	#[attributes.requestUri]
Maximum Cache Entries	10000
Entry Time To Live (in Seconds)	600
Distributed	true
Persistent Cache	true
Follow HTTP Caching directives	true
Conditional Request Caching Expression	#[attributes.method == 'GET' or attributes.method == 'HEAD']
Conditional Response Caching Expression	#[[200] contains attributes.statusCode]

Invalidation Header

Name of the header to be used for the cache invalidation. To invalidate an entry send a request with this header with one of the following values: invalidate, invalidate-all. Invalidate-all invalidates all the key-value pairs from the cache. For security reason this header is always masked.

••••••••••• **x-cache-invalid-header**

```
curl http://app1.us-e2.cloudhub.com/api/users/123 -H "x-cache-invalid-header:invalidate"
```

```
curl http://app1.us-e2.cloudhub.com/api/users/456 -H "x-cache-invalid-header:invalidate-all"
```

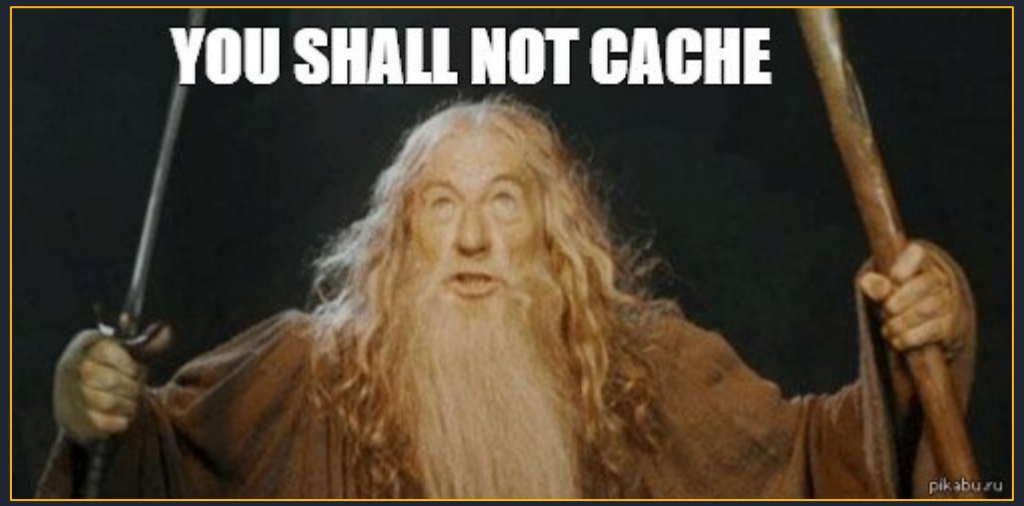
Internally uses Mule's Object Store to store cache

Follows HTTP caching directives including **invalidation**

Object Store v2 must be enabled on the application to use persistent caching for the HTTP caching API policy

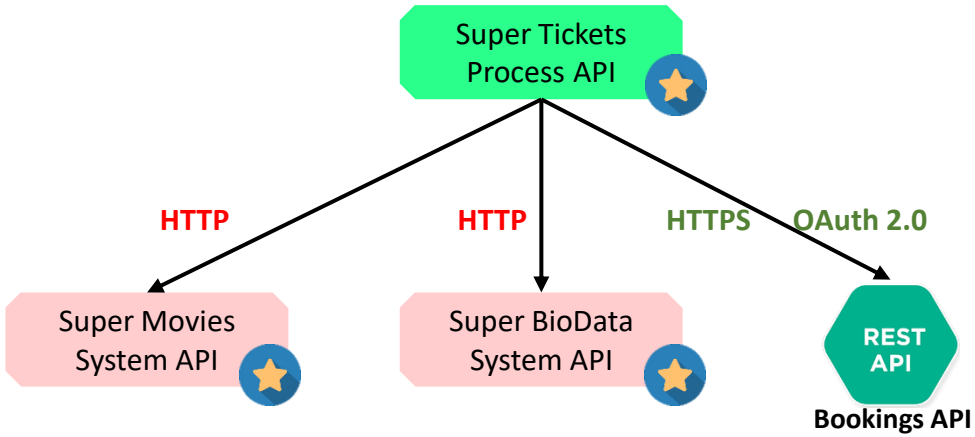
2.13 Sever-Side Caching

- API Caching Policy
- Invalidate Header
- Visulaizer



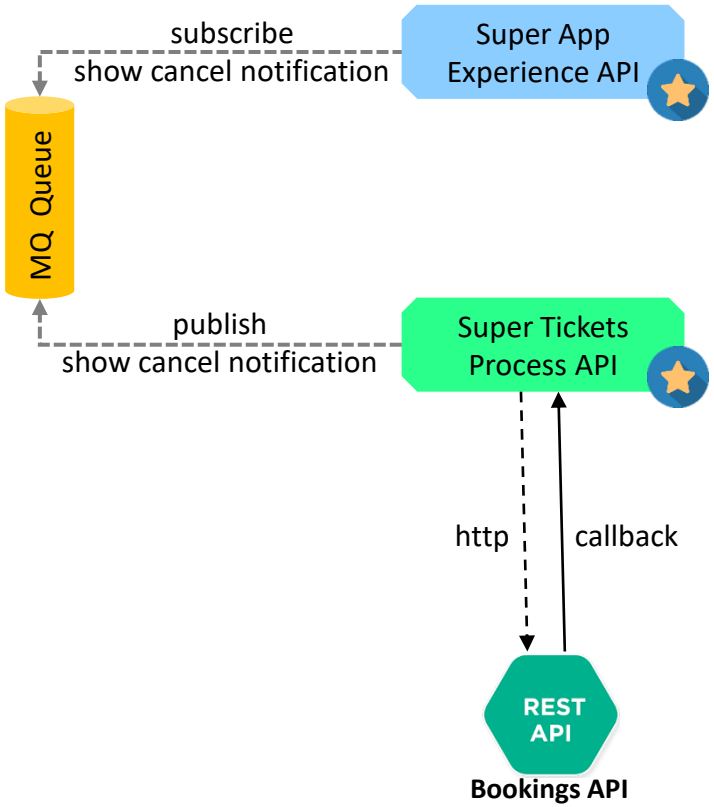
#2 Mule Integration - API Development, Error Handling, Caching Strategies

- Add System API REST Connectors to Anypoint Studio
- Get credentials and invoke the SAPI
- Configure HTTP Non-functional requirement
- OAuth 2.0 – Client Credentials Grant Type
- Add DEBUG Async Logging
- Add transformation logic
- Handle custom exceptions
- Parallel execution using Scatter-gather
- Scatter-gather error handling
- Use Until Successful scope to handle HTTP Errors
- Transient vs Permanent HTTP Errors
- Client Side Caching using Object Store/Cache Scope
- Server Side Caching using API Manager's API Caching policy



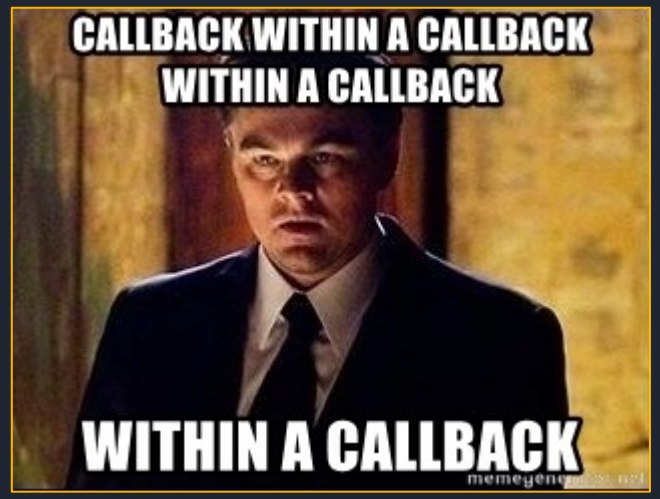
#3 Asynchronous Message Processing, Logging Options, Tracing, Correlation IDs and Content Validation

- Configure HTTP Callback
- Register HTTP Callback with external service
- Configure VM module and add Publish Messages
- Listen to messages by adding a Transaction
- Add redelivery policy using correlation id
- Forward error messages to DLQ
- Configure Anypoint MQ module
- Publish messages to MQ
- Create new App and subscribe Anypoint MQ messages
- Publish messages to external system
- Configure a circuit breaker
- Check Correlation ID across HTTP and VM protocols
- Trace correlation id using MQ publish operation
- Operational logging and tracing
- Use Mapped Diagnostic Context
- Configure Validation Module
- Validate Input request
- Validate XML/JSON Schemas

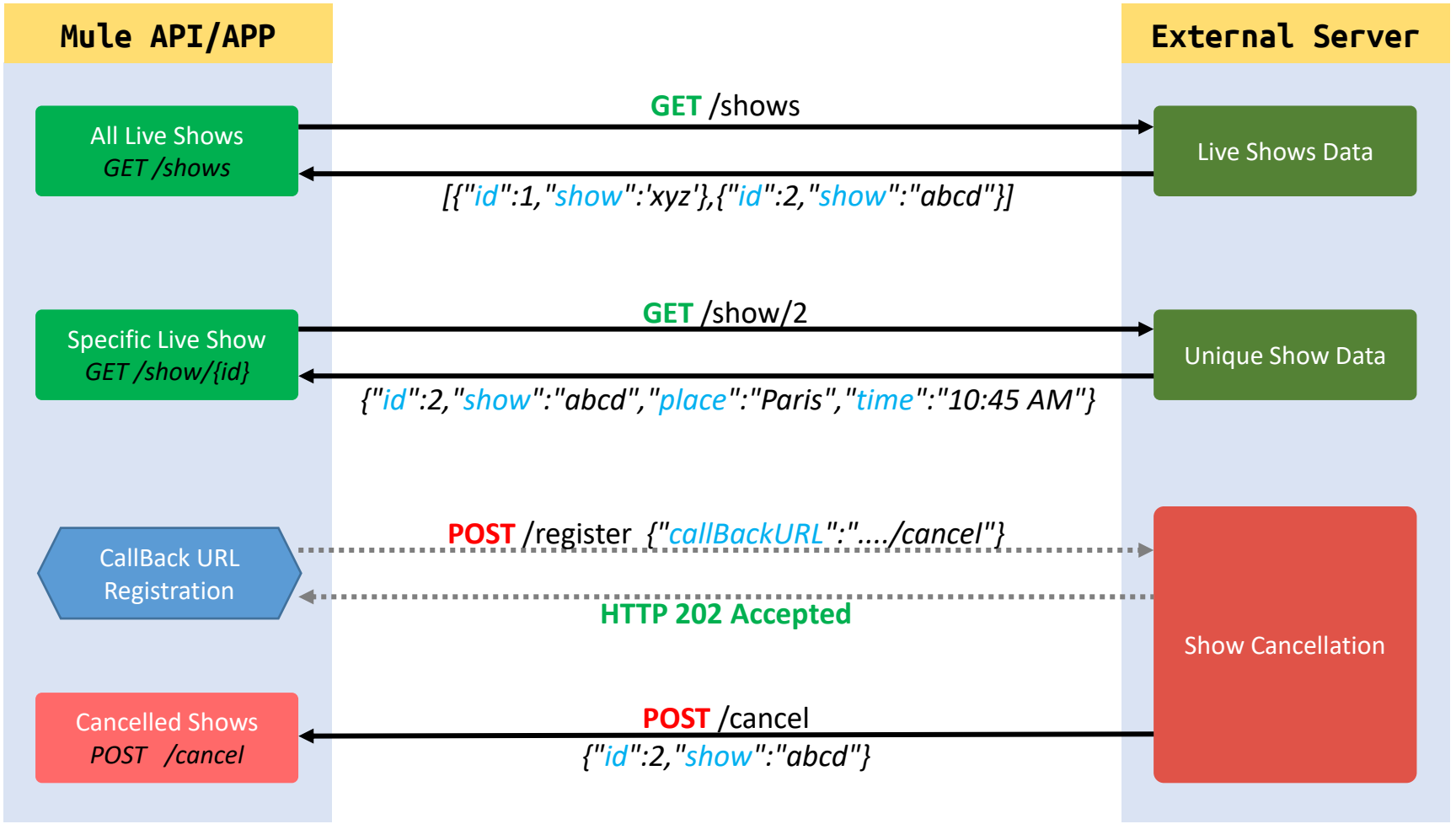


HTTP Callback

- What/Why?



HTTP Callbacks/Webhooks



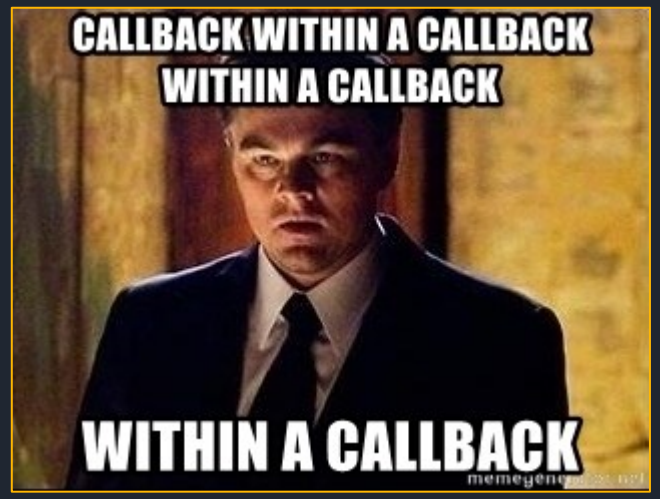
Mule App creates an flow with HTTP Listener (typically **POST** method) to receive cancelled show data

Mule App registers/sends the endpoint URL to the external server

External Server sends HTTP POST requests to that **Callback URL** whenever a Show is cancelled

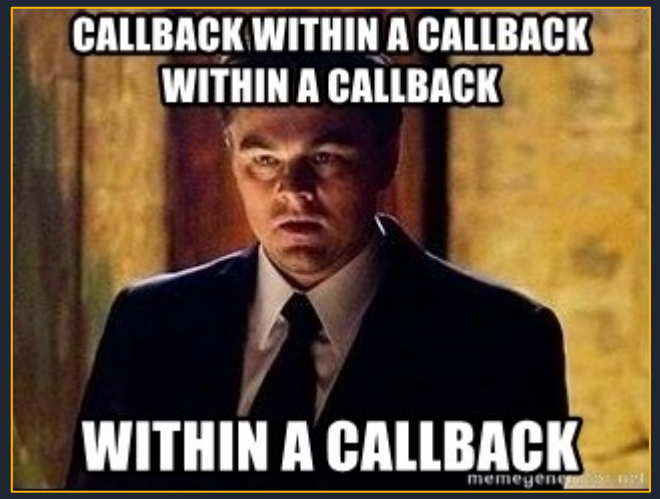
3.1 HTTP Callback

- Configure in Mule Application



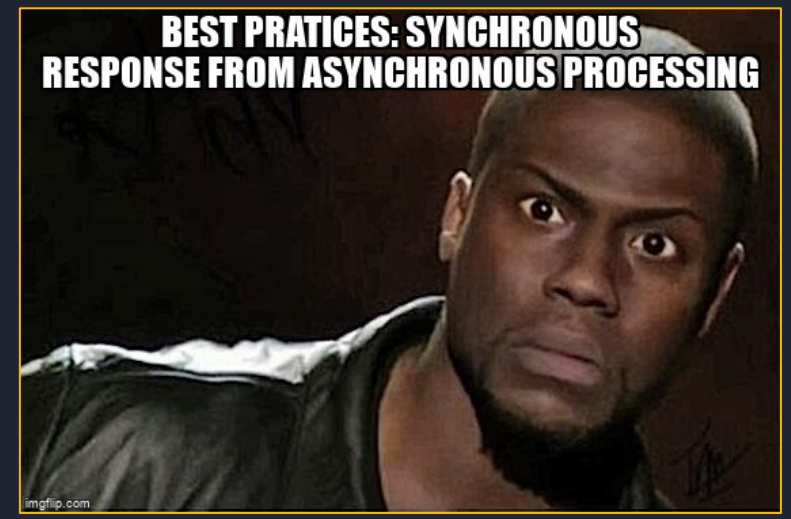
3.2 HTTP Callback

- Register Callback with external service
- CloudHub Reserved Properties

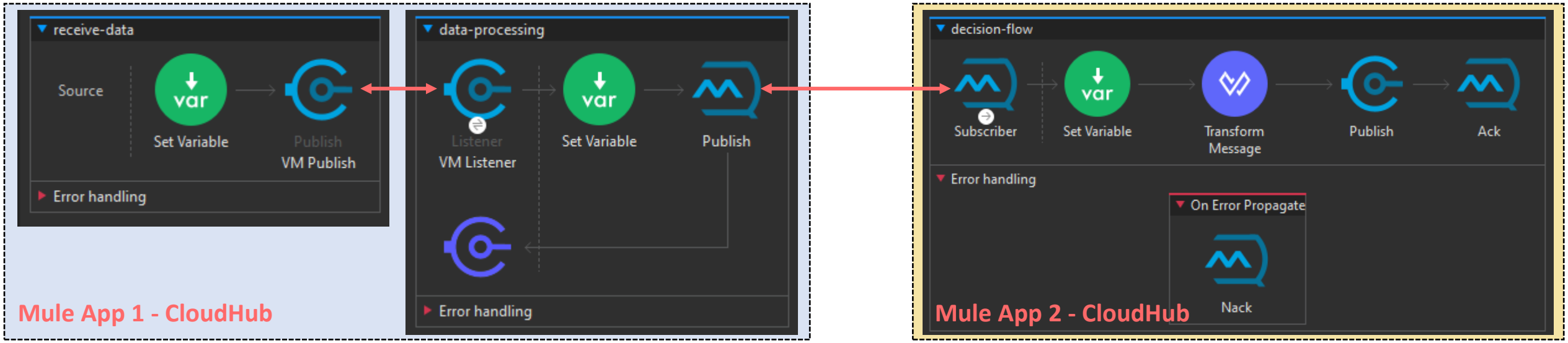


Asynchronous Processing

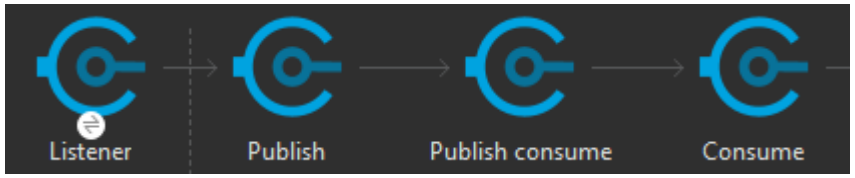
- What/Why?
- VM vs Anypoint MQ



Asynchronous Message Processing



- Use the **VM connector** to pass messages **between flows** using asynchronous queues
  - **Transient queues** are faster, but are lost in the case of a app crash
  - **Persistent queues** are slower but reliable
    - CloudHub Persistent Queues uses **Amazon SQS**
  - **VM:Publish** is one-way operation – fire & forget
  - **VM:Publish consume** is blocking request/response operation
  - Achieve message reliability through [Reliability patterns]
  - Send events across different apps (apps in **same domain** (on-prem))



- Use **Anypoint MQ** connector to pass messages between flows and Mule/Non-Mule applications
  - Fully managed and Cloud hosted
    - URL | Client ID | Client Secret
  - Simple Queueing, Pub/Sub using Exchanges, FIFO
  - Text, JSON or CSV message format
  - **Circuit breaker** pattern to handle failing messages
  - **ACK Modes –** Auto | Manual | IMMEDIATE

VM Queues

Queue	staging area that contains messages that have been sent and are waiting to be read
DLQ	the same as any other queue except that it receives only undelivered messages

```xml
<vm:config name="vmConfig">
    <vm:queues>
        <vm:queue
                queueName="example-q"
                queueType="PERSISTENT"
                maxOutstandingMessages="10" />
        <vm:queue
                queueName="example-dlq"
                queueType="TRANSIENT"
                maxOutstandingMessages="100" />
    </vm:queues>
</vm:config>
```

```xml
<vm:publish
    config-ref="vmConfig"
    queueName="example-q"
    timeout="5000"
    timeoutUnit="MILLISECONDS"
    sendCorrelationId="ALWAYS" />
```

```xml
<vm:listener
        config-ref="vmConfig"
        queueName="example-q"
        numberOfConsumers="2"
        timeout="5000"
        timeoutUnit="MILLISECONDS"
        transactionalAction="ALWAYS_BEGIN">
    <redelivery-policy
        idExpression="#[correlationId]"
        maxRedeliveryCount="4" />
</vm:listener>
```

Reliability pattern also strongly suggests that the queues should be persistent

Limit Queue Size - in the case of a DoS attack, if messages are dequeued slower than they are enqueued then the Mule app will fail with an out-of-memory error

Explicit timeout ensures that messages sent to the queue does not block indefinitely in case of an error

The HTTP Listener generates the correlationId and it will be used within VM connector as well

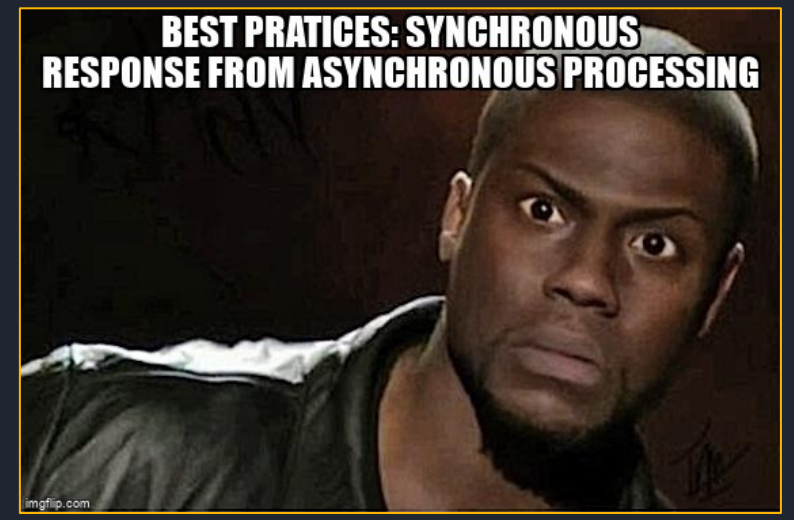
The transaction starts with the de-queuing of the message (is a single VM resource-local transaction)

Redelivery Policy uses correlationId to identify and redelivery the dequeued message and internally uses a Default Object Store

Redelivery Policy raises a MULE:REDELIVERY_EXHAUSTED error, which needs to be caught and send message to DLQ

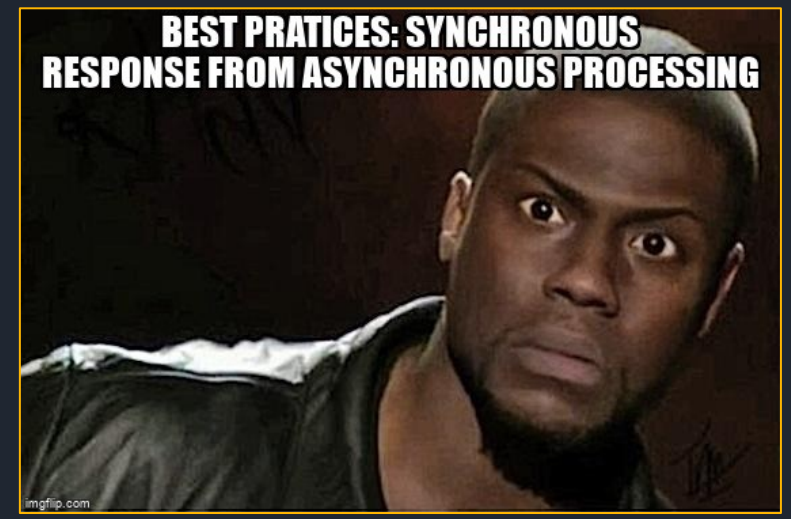
3.3 Asynchronous Processing

- Reliability Pattern
- VM Configuration
- VM Connector
    - Publish Payloads



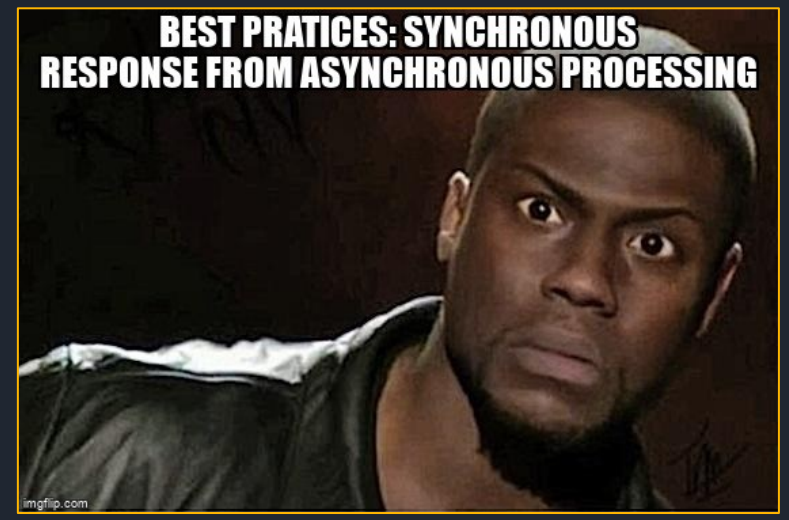
3.4 Asynchronous Processing

- VM Connector
  - Listen for Payloads
  - Add transaction
  - Check for error



3.5 Asynchronous Processing

- VM Connector
  - Re-delivery Policy



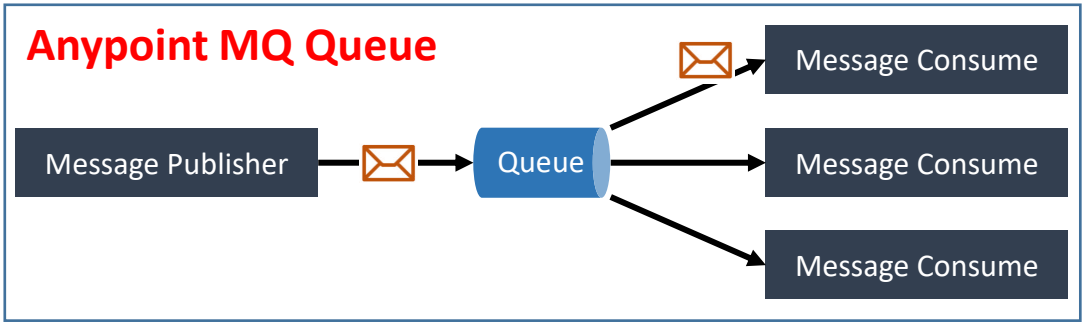
3.6 Asynchronous Processing

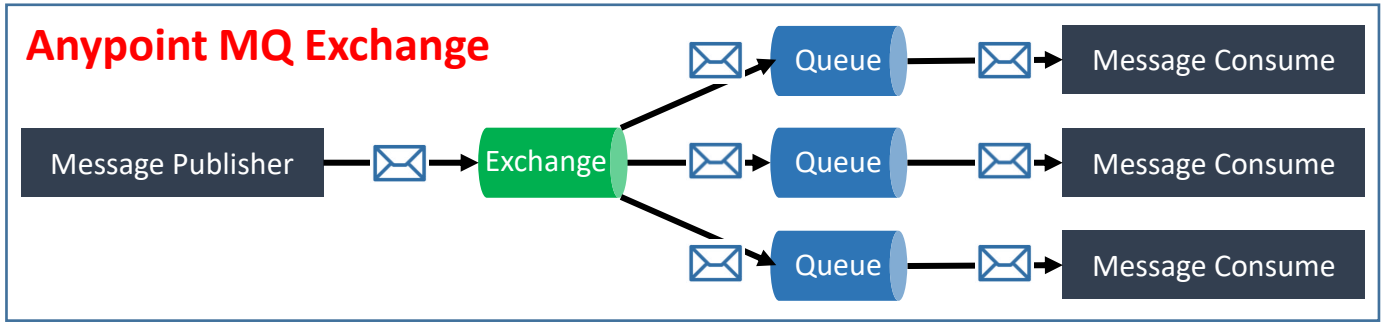
- VM Connector
  - Add a DLQ
  - Error Handling

Anypoint MQ

- What/How?
- Queue vs Exchange
- Steps

Anypoint MQ



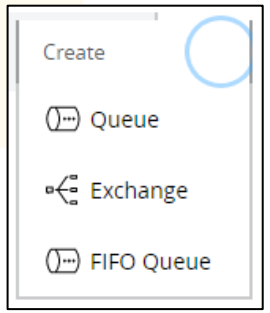


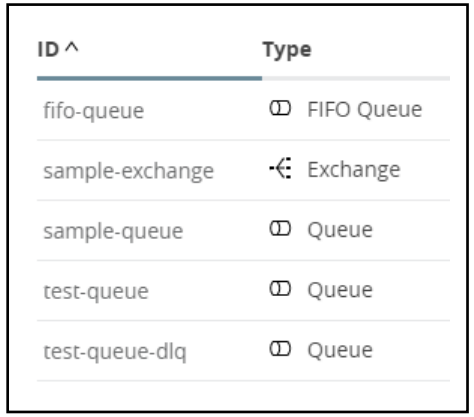
Anypoint MQ Queue

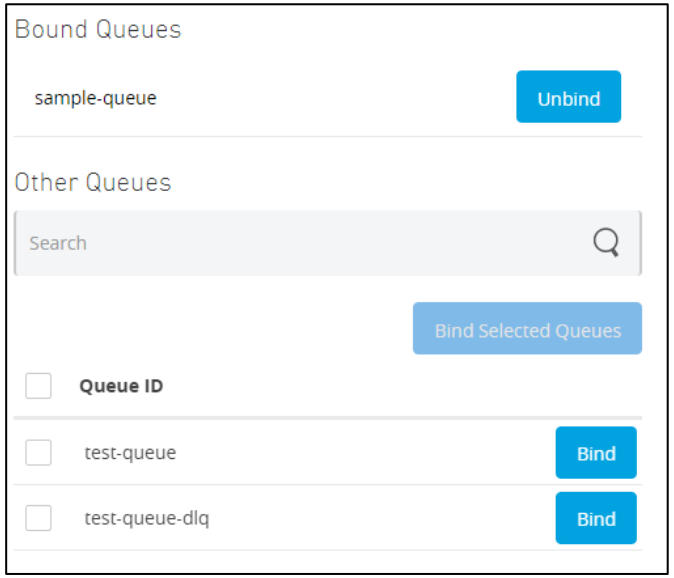
- **Queue** (Standard Queue)
  - Send & receive messages unordered (does not guarantee order)
  - Best fit for fast delivery, but only when order is not important
- **FIFO** (First in First Out)
  - Ordering of messaging is maintained
  - Built-in deduplication of messages (Exactly Once Delivery)
- **DLQ** (Dead Letter Queue)
  - Same queue type, region, encryption as origin queue
  - 10 reroute attempts before sending to DLQ
  - Recovery of message in DLQ via MQ Administration API
  - Can be associated with one or more queues
- Message **Format** & Size
  - Text, JSON or CSV format message
  - Maximum 10MB message size

Anypoint MQ Exchange

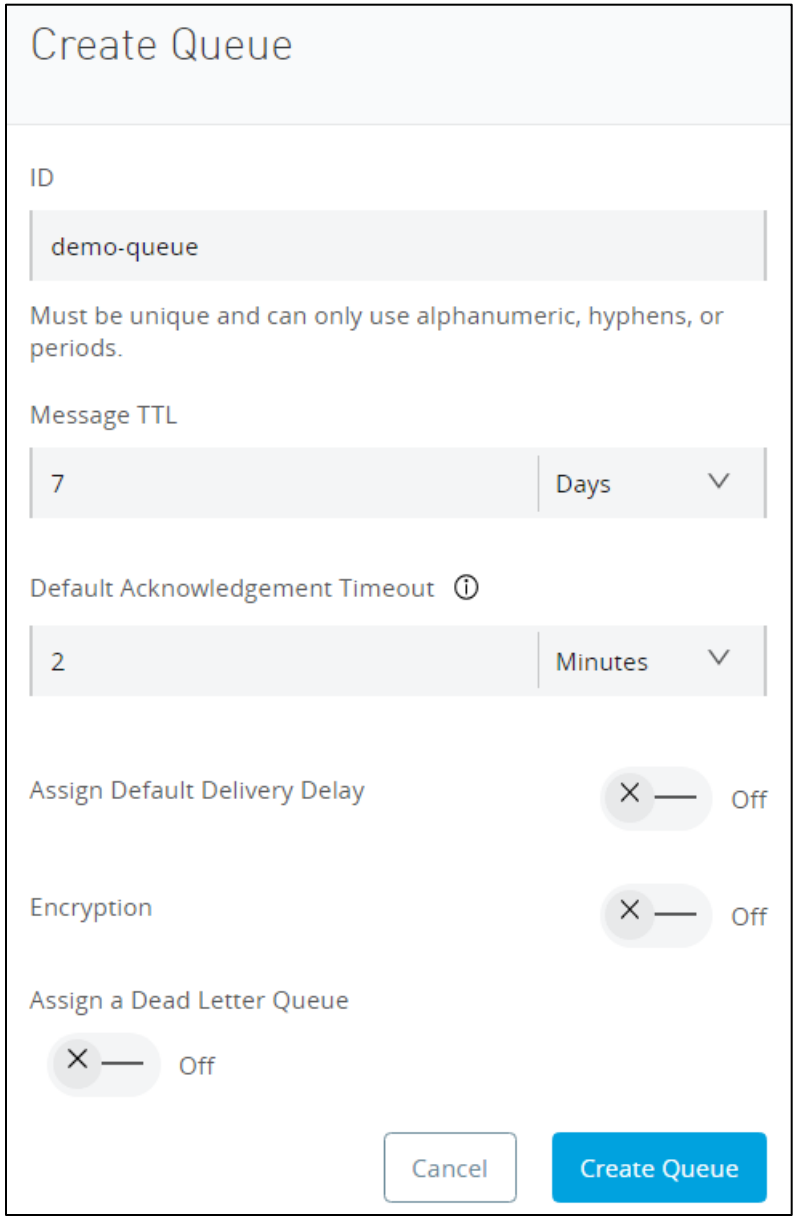
- Used to Broadcasting messages to Queues
- Bind one or more Queues to Exchange
  - FIFO queues are not supported

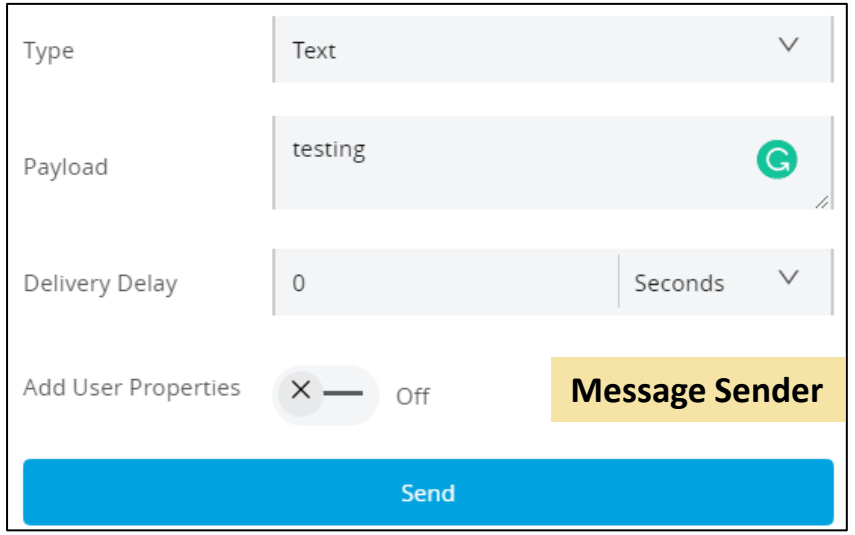


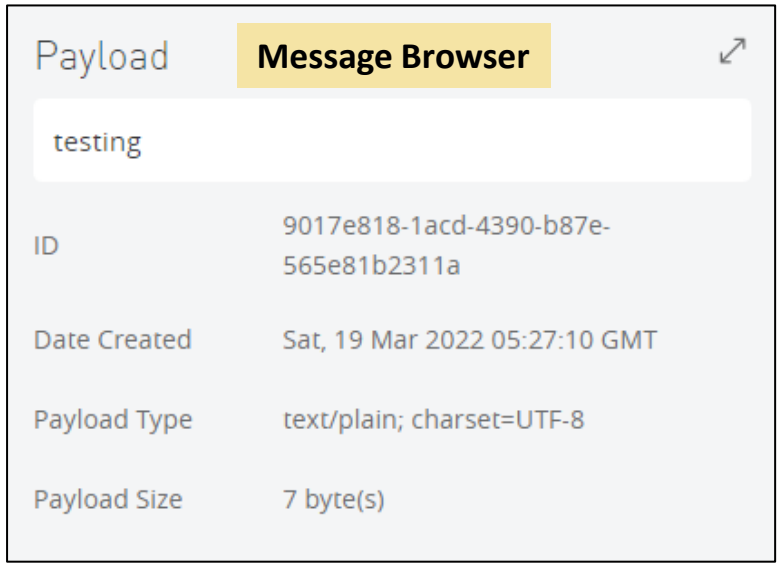


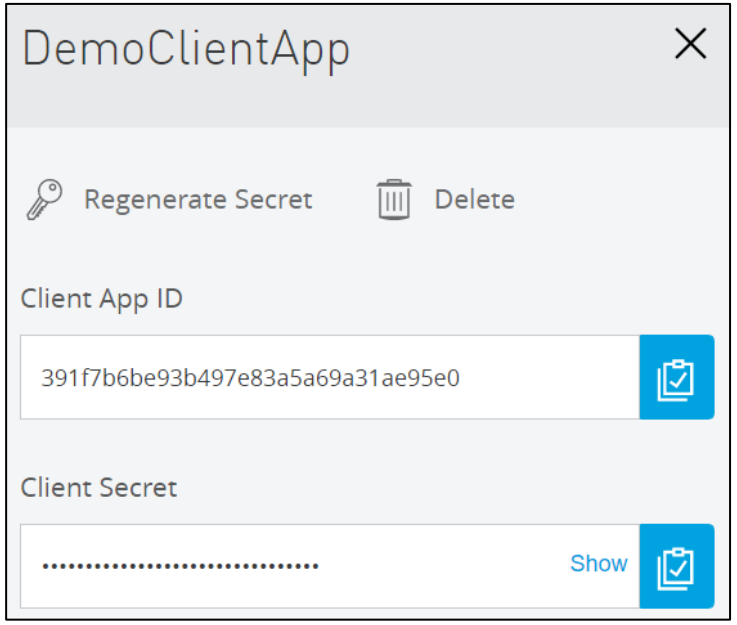


Anypoint MQ - Steps









- Clients have the following attributes,
  - name
  - client id
  - client secret
- Clients cannot access queues across envs
  - Each consume should have a client

```
<anypoint-mq:config name="anypointMQConfig">
    <anypoint-mq:connection url="${secure::mq.url}"
        clientId="${secure::mq.client_id}"
        clientSecret="${secure::mq.client_secret}" />
</anypoint-mq:config>
```

3.7 Anypoint MQ

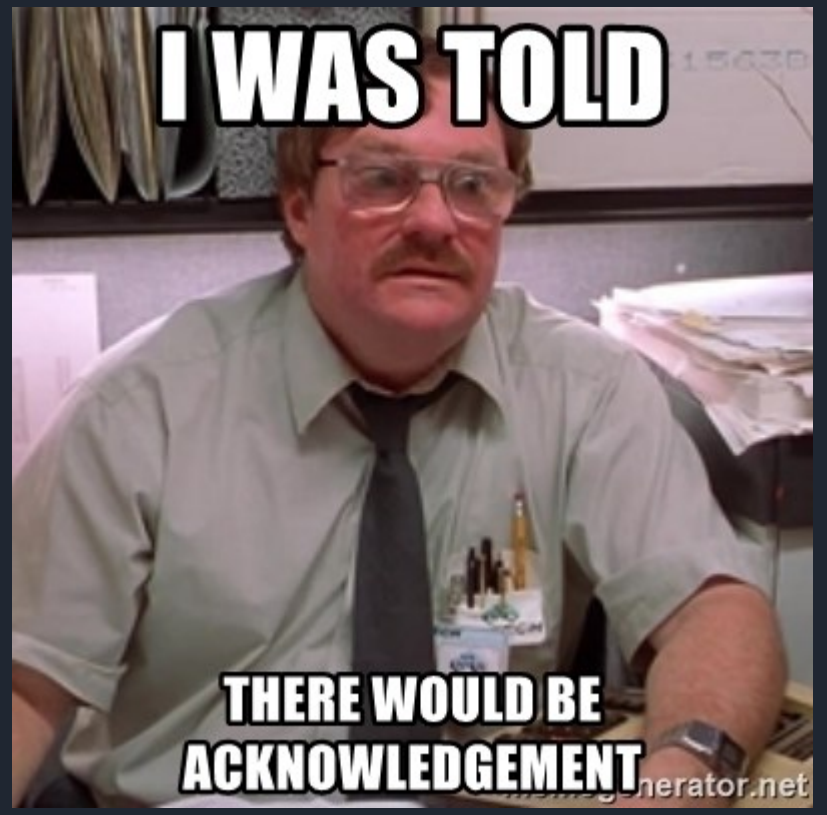
- Configure Anypoint MQ in Mule app

3.8 Anypoint MQ

- Publish Messages to Anypoint MQ

Anypoint MQ

- Message Acknowledgement
  - Acknowledgement Types
    - ACK
    - NACK
  - Acknowledgement Modes
    - AUTO
    - IMMEDIATE
    - MANUAL



Message Acknowledgement

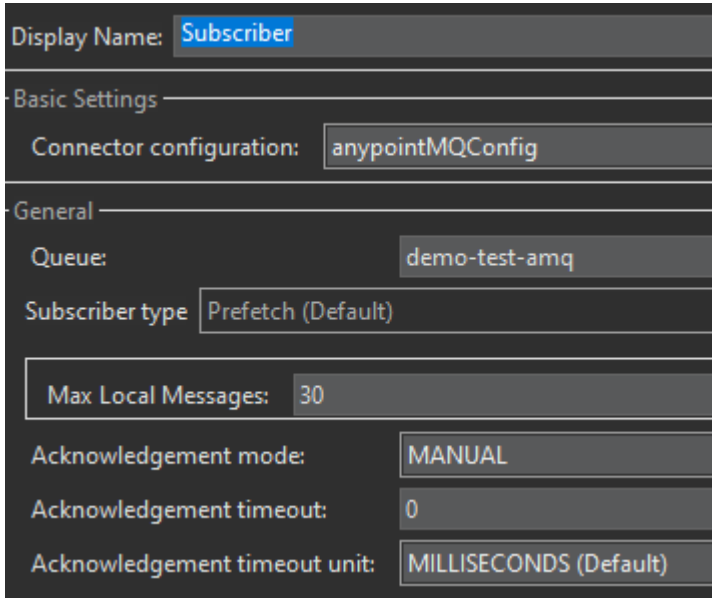
Acknowledgement Types

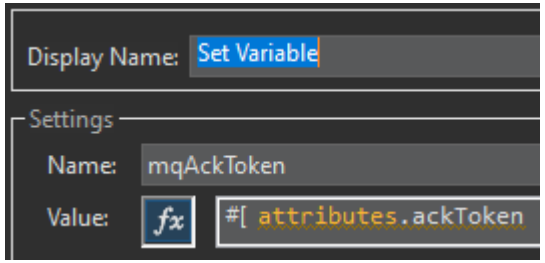
- **ACK**
  - Indicates that an application has processed the message
  - Messages will be **deleted** from Queue
- **NACK**
  - Indicates that an application did not process the message
  - Messages **remain** in Queue

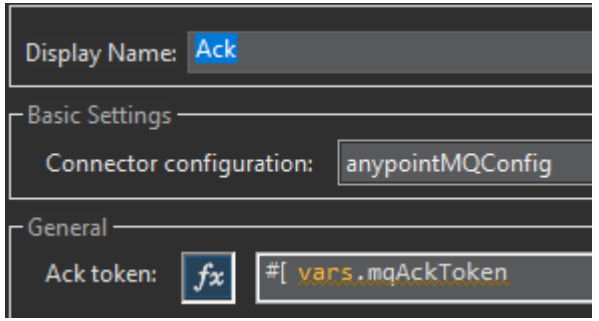
- **acknowledgementTimeout (optional)**
  - time that Anypoint MQ waits to receive an ACK or NACK
  - default TTL is 2 minutes
  - Applicable for,
    - MANUAL or AUTO acknowledgment modes

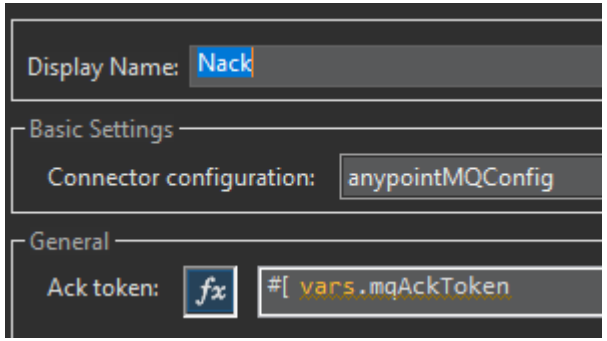
Acknowledgement Modes

- **AUTO** (default)
  - Sends ACK only if flow execution is successfully
  - Sends NACK if case of error, message returned to queue
  - Only applicable for **Subscriber** message source
- **MANUAL**
  - Responsibility of application logic to manage ACK/NACK
  - requires a **AnypointMQMessageContext** object
- **IMMEDIATE**
  - Prior to processing, directly sends ACK
  - Message gets removed from queue







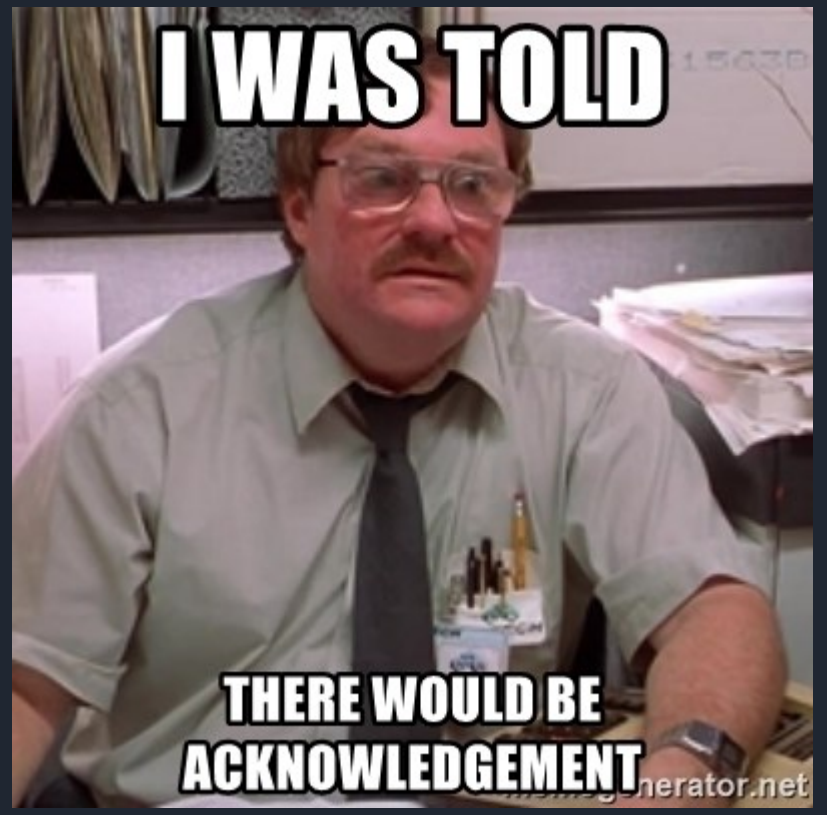


Acknowledgement Token (ackToken)

ackToken is a unique identifier for the message that must be used when executing **MANUAL** ACK/NACK **operation**

3.9 Anypoint MQ

- Create new EAPI
- Subscribe to Anypoint MQ Queue
- Manual ACK

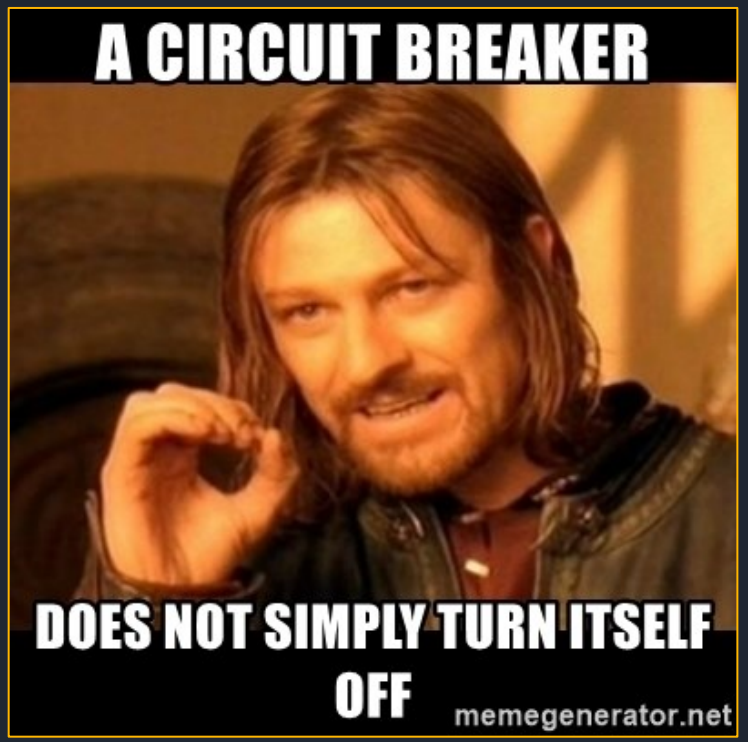


3.10 Anypoint MQ

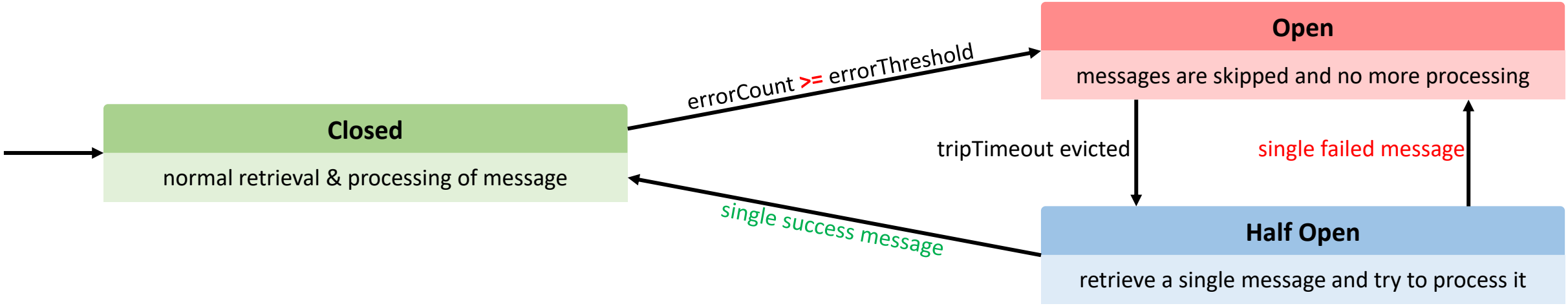
- Publish events to external system
- Manually Raise an error
- Handle error using NACK

Anypoint MQ

- Circuit Breaker
  - Configuration



Anypoint MQ – Circuit Breaker




```
35  <anypoint-mq:circuit-breaker
36      name="apmqCircuitBreaker"
37      tripTimeout="20"
38      onErrorTypes="HTTP:CONNECTIVITY"
39      errorsThreshold="1"
40      tripTimeoutUnit="SECONDS" />
```

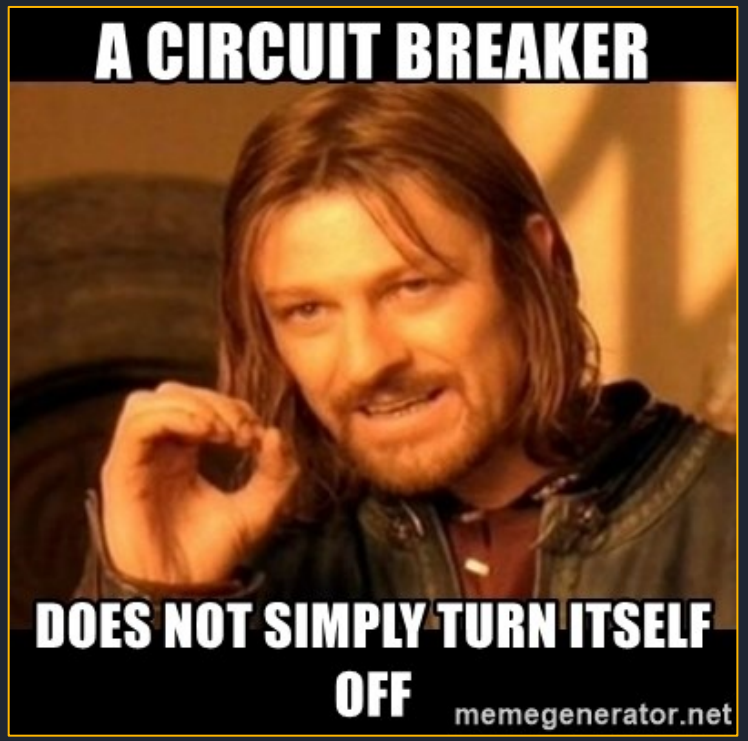
```
47  <anypoint-mq:subscriber
48      config-ref="anypointMQConfig"
49      destination="${secure::mq.destination}"
50      acknowledgementMode="MANUAL"
51      circuitBreaker="apmqCircuitBreaker"/>
```

Circuit Breaker ONLY applicable to - anypoint-mq:subscriber

onErrorTypes	errorsThreshold
error types that count as a failure during the flow execution	number of onErrorTypes errors that must occur for the circuit breaker to open
tripTimeout	**circuitName**
how long the circuit remains open once errorsThreshold is reached	circuit breaker to bind to this configuration. Each queue has its own circuit breaker

3.11 Anypoint MQ

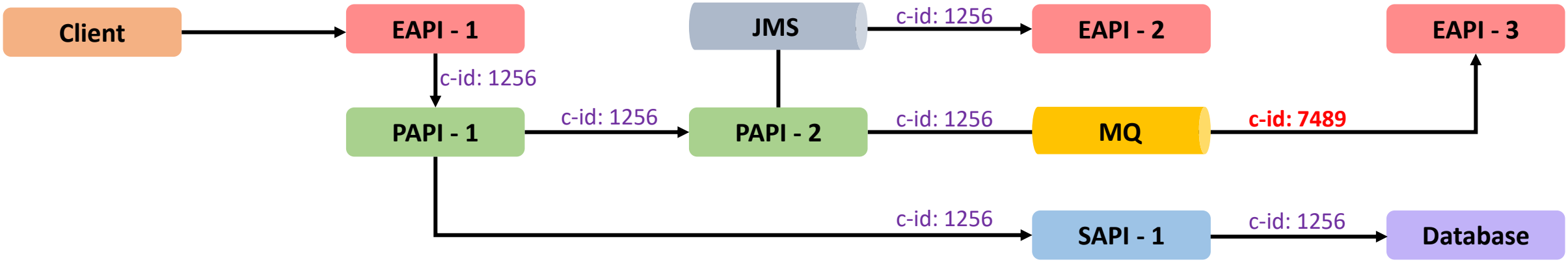
- Circuit Breaker



Correlation ID

- What?
- How?

Correlation ID

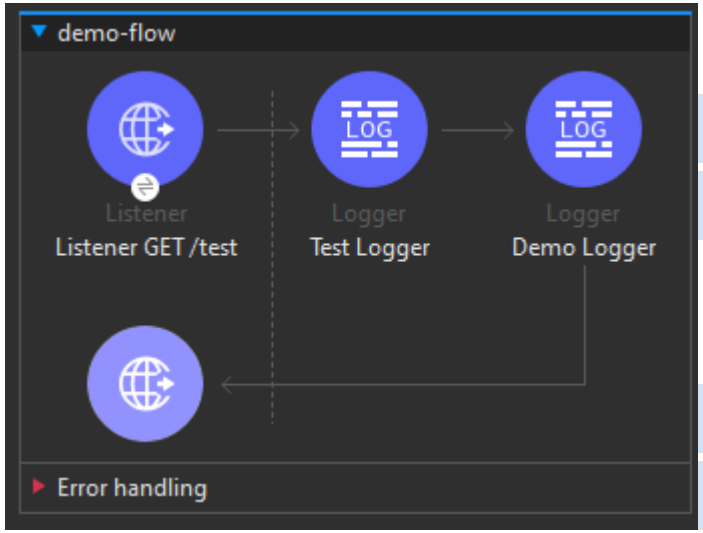


Correlation IDs are immutable by design and cannot be changed within a flow *(needs additional **configuration/modules**)*

Transport connectors like **HTTP, JMS, VM** automatically send the correlation ID by default *(clients can set custom id by using **X-CORRELATION-ID** header)*

Anypoint MQ connector does not automatically propagate correlation ID. The Anypoint MQ **messageId** has no effect on the Mule message **correlation ID**

Anypoint MQ always regenerates the correlation ID *(use a **custom property + Tracing module** for propagating the correlation ID)*




```
curl http://localhost:8081/test
```

INFO … … … [processor: demo-flow/processors/0; event: ed275080-a7a2-11ec-8535-1c1b0d68700c] … … : Test Logger

INFO … … … [processor: demo-flow/processors/1; event: ed275080-a7a2-11ec-8535-1c1b0d68700c] … : Demo Logger

```
curl http://localhost:8081/test -H "X-CORRELATION-ID:11111-222222-333333-44444-55555"
```

INFO … … … [processor: demo-flow/processors/0; event: 11111-222222-333333-44444-55555] … … : Test Logger

INFO … … … [processor: demo-flow/processors/1; event: 11111-222222-333333-44444-55555] … : Demo Logger

3.12 Correlation ID

- Trace correlation ID
  - across HTTP Listener
  - across VM
  - across Anypoint MQ

Logging & Tracing

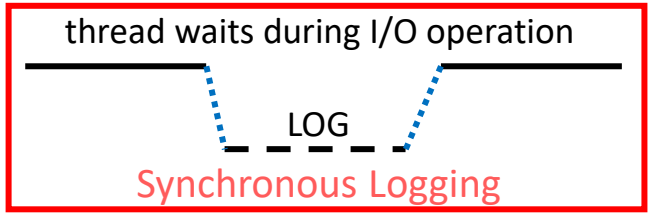
- What/Why/How?
- Log
  - PatternLayout
- Tracing Module
    - MDC

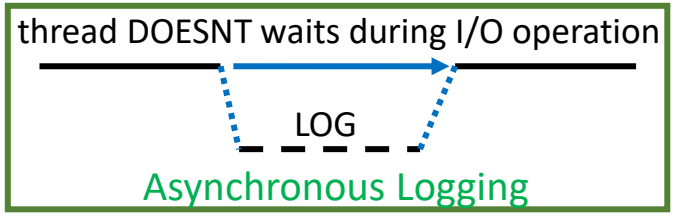
Logging & Tracing Module

Log Level - **INFO** *(ignores DEBUG or TRACE level log)* **&** Log messages can be configured to log to various **appenders** *(CloudHub, console, file, database)*

Default is **asynchronous** logging, supports synchronous(MUnit) logging *(configured with standard log4j2.xml)*

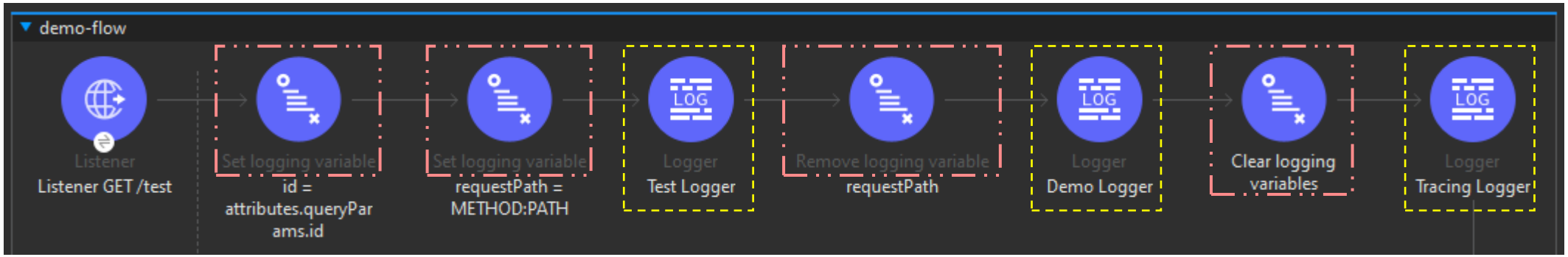
CloudHub **ignores** and **overrides** the application's **log4j2** configuration *(optionally we can **Disable** CloudHub logs and log to external systems as well)*

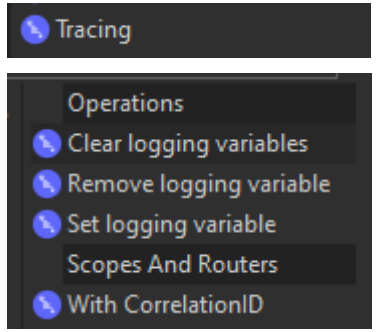




<**PatternLayout** pattern="%-5p %d [%t] [**processor: %X{processorPath}**; **event: %X{correlationId}**] %c: %m%n"/> ← **OOTB Pattern Layout**

<**PatternLayout** pattern="%-5p %d [%t] **[%MDC]** %c: %m%n"/> ← **Mapped Diagnostic Context (MDC)**






```
curl -X GET http://localhost:8081/test?id=007
```

INFO … … … … … [{correlationId=21f578d0-a7b4-11ec-9042-1c1b0d68700c, **id=007**, processorPath=demo-flow/processors/2, **requestPath=GET:/test**}] … : Test Logger

INFO … … … … … [{correlationId=21f578d0-a7b4-11ec-9042-1c1b0d68700c, **id=007**, processorPath=demo-flow/processors/4] … : Demo Logger

INFO … … … … … [{correlationId=21f578d0-a7b4-11ec-9042-1c1b0d68700c, processorPath=demo-flow/processors/6}] … : Tracing Logger

3.13 Logging & Tracing

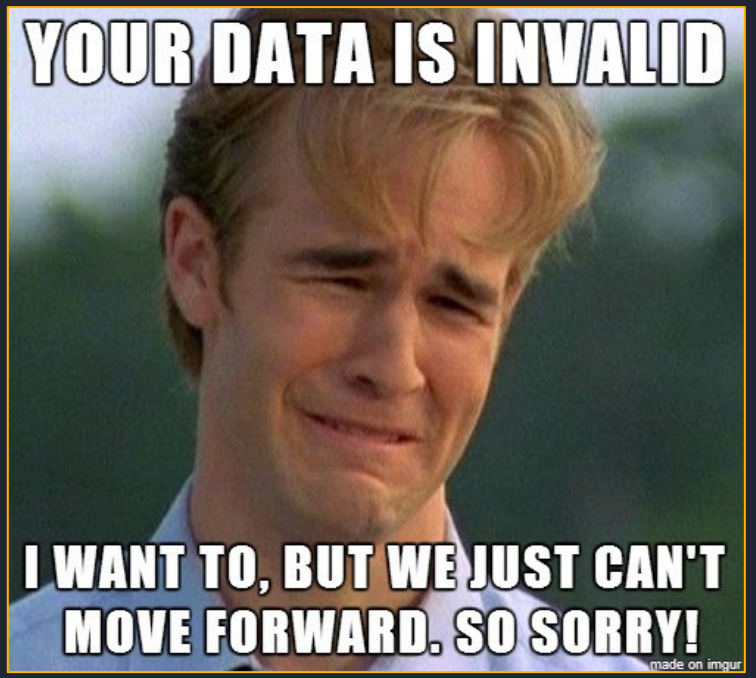
- Tracing Correlation ID across Anypoint MQ

3.14 Logging & Tracing

- Mapped Diagnostic Context

Validation Module

- What/Why/How?
- JSON Schema Validation
- XML Schema Validation

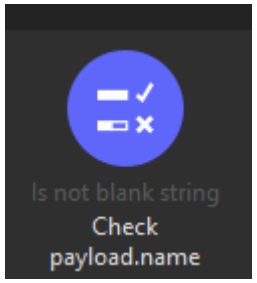


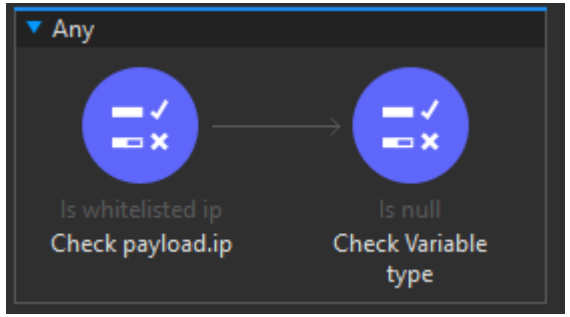
Validation Module

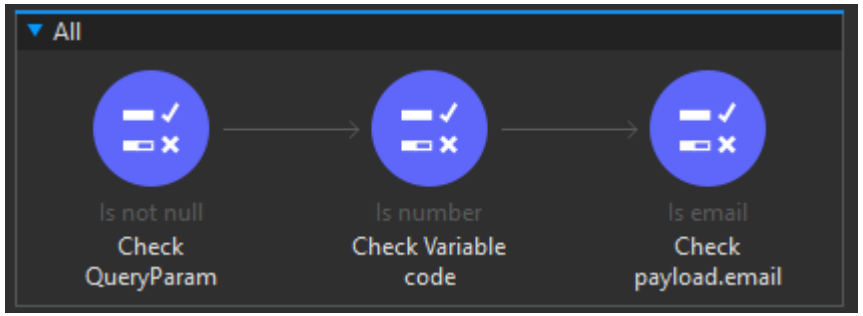
For APIs, **APIKit router** validates requests with API Specifications (DataTypes/Schemas) **in RAML/OAS/WSDL**

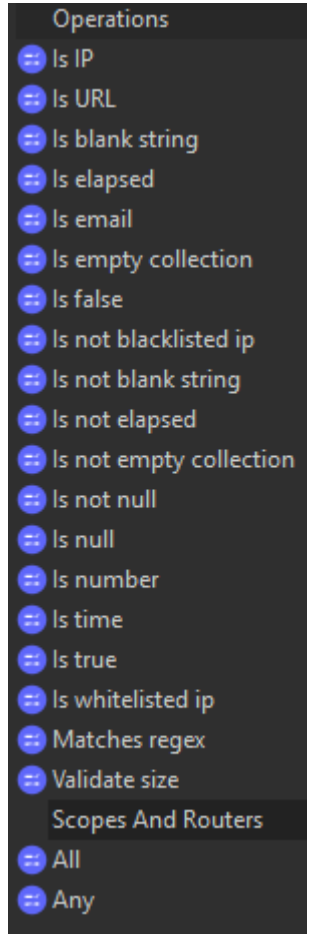
Validation Module is used to validate messages of **VM/JMS/MQ** endpoints *(Async API will be the future)*

Used for pre/post conditions validation and Raises a **VALIDATION:*** error type










```
3   <validation:is-email  email="#[payload.email]" message="Validation failed, received an invalid email.">
4                   <error-mapping sourceType="VALIDATION:INVALID_EMAIL" targetType="APP:INVALID_EMAIL" />
5   </validation:is-email>
```

XML & JSON Schema Validation

https://martinfowler.com/bliki/TolerantReader.html

MCD2

Robustness Principle - *(Postel's Law)*

be conservative in what you do, be liberal in what you accept from others

Integration Pattern *(Tolerant Reader)* – idea is to be as tolerant as possible when reading data from another service

```
3  <xml-module:validate-schema  schemas="schemas/XmlDemoSchema.xsd" />
```

schemas/XmlDemoSchema.xsd

```
17  <xs:schema
18      attributeFormDefault="unqualified"
19      elementFormDefault="qualified"
20      xmlns:xs="http://www.w3.org/2001/XMLSchema">
21      <xs:element name="root">
22          <xs:complexType>
23              <xs:sequence>
24                  <xs:element type="xs:string" name="title" />
25                  <xs:element type="xs:string" name="tool" />
25                  <xs:any processContents="lax" minOccurs="0" />
27              </xs:sequence>
28          </xs:complexType>
29      </xs:element>
30  </xs:schema>
```

lax - if the schema cannot be obtained, no errors will occur

```
33  <root>
34    <title>XML Validation Demo</title>
35    <tool>MuleSoft</tool>
36
37  </root>
```

```
39  <root>
40    <title>XML Validation Demo</title>
41    <tool>MuleSoft</tool>
42    <version>4.4.0</version>
43  </root>
```

Message : Input **XML** was not compliant with the schema.

Error type : **XML-MODULE**:SCHEMA_NOT_HONOURED

```
1  <json:validate-schema  schema="schemas/JsonDemoSchema.json" />
```

schemas/JsonDemoSchema.json

```
1   {
2       "$schema": "http://json-schema.org/draft-07/schema",
3       "$id": "http://example.com/example.json",
4       "type": "object",
5       "examples": [ ],
6       "required": [
7           "title", "tool"
8           ],
9       "properties": {
10          "title": { },
11          "tool": { }
12      },
13      "additionalProperties": true
14  }
```

additionalProperties handles extra fields i.e not listed in properties keyword

```
1  {
2    "title": "JSON  Validation",
3    "tool": "MuleSoft"
4
5  }
```

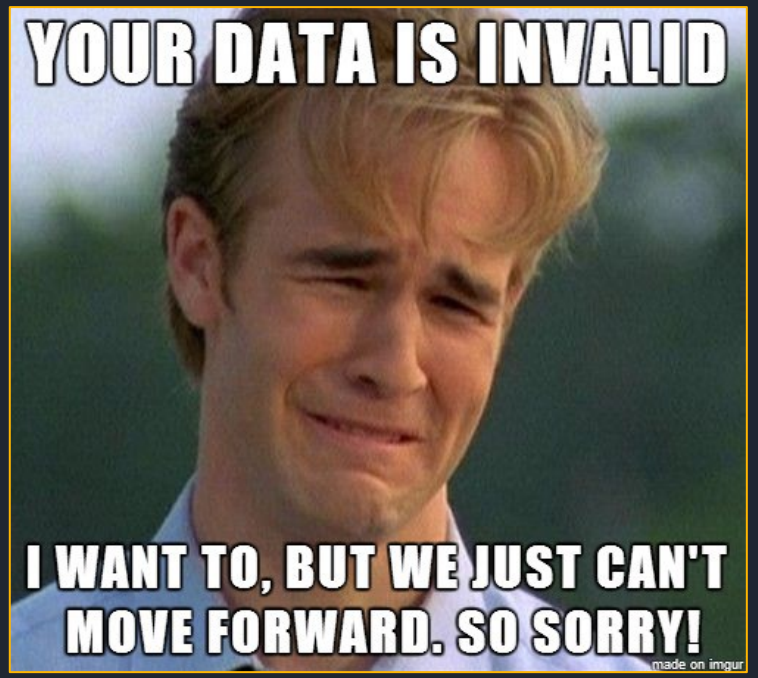
```
1  {
2    "title": "JSON  Validation",
3    "tool": "MuleSoft",
4    "version": "4.4.0"
5  }
```

Message : **Json** content is not compliant with schema.

Error type : **JSON**:SCHEMA_NOT_HONOURED

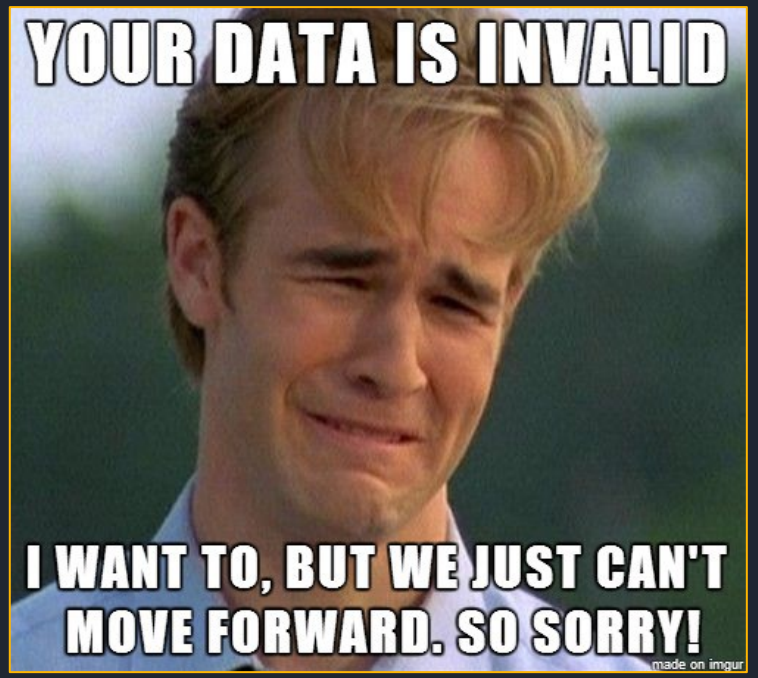
3.15 Validation Module

- Configure module
- Validate Payload and Content type
- Handle Errors
- Test



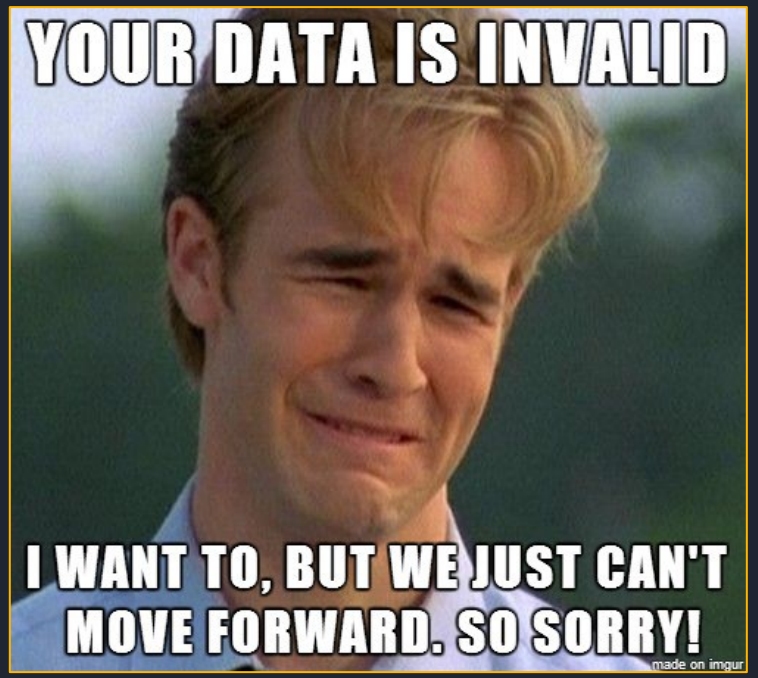
3.16 JSON Schema Validation

- Configure module
- Validate JSON Payload
- Add additional properties
- Handle Errors
- Test



3.17 XML Schema Validation

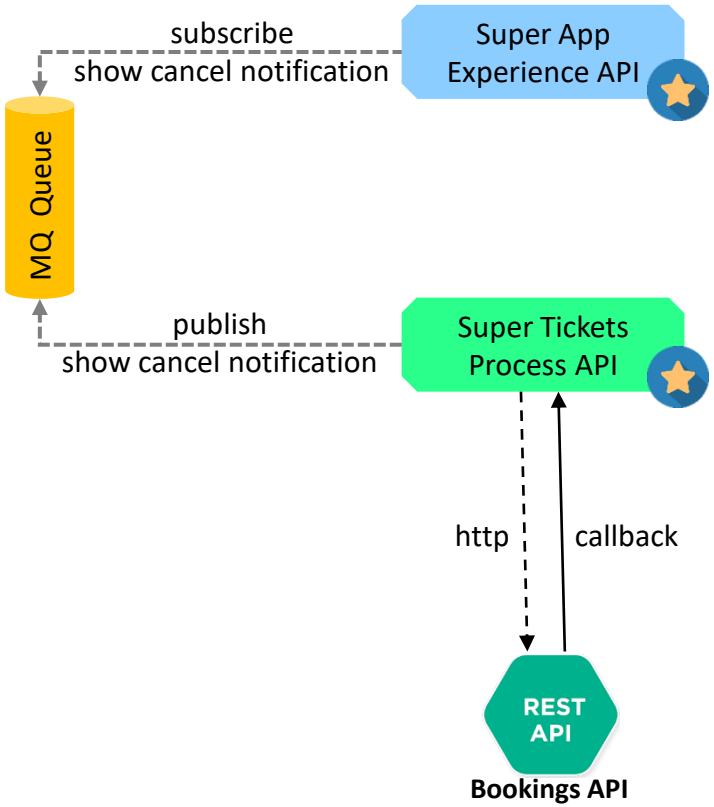
- Configure module
- Validate XML Payload
- Add lax
- Handle Errors
- Test



Section 3 - Completed

#3 Asynchronous Message Processing, Logging Options, Tracing, Correlation IDs and Content Validation

- Configure HTTP Callback
- Register HTTP Callback with external service
- Configure VM module and add Publish Messages
- Listen to messages by adding a Transaction
- Add redelivery policy using correlation id
- Forward error messages to DLQ
- Configure Anypoint MQ module
- Publish messages to MQ
- Create new App and subscribe Anypoint MQ messages
- Publish messages to external system
- Configure a circuit breaker
- Check Correlation ID across HTTP and VM protocols
- Trace correlation id using MQ publish operation
- Operational logging and tracing
- Use Mapped Diagnostic Context
- Configure Validation Module
- Validate Input request
- Validate XML/JSON Schemas



#4 Mule Application Health Checks and Custom Connector Development

- Configure health endpoint
- Understand Liveness vs Readiness concepts
- Create a new library with common code
- Install it to local maven repo
- Create a new connector using XML SDK
- Build, Package and Publish it to Exchange
- API Monitoring

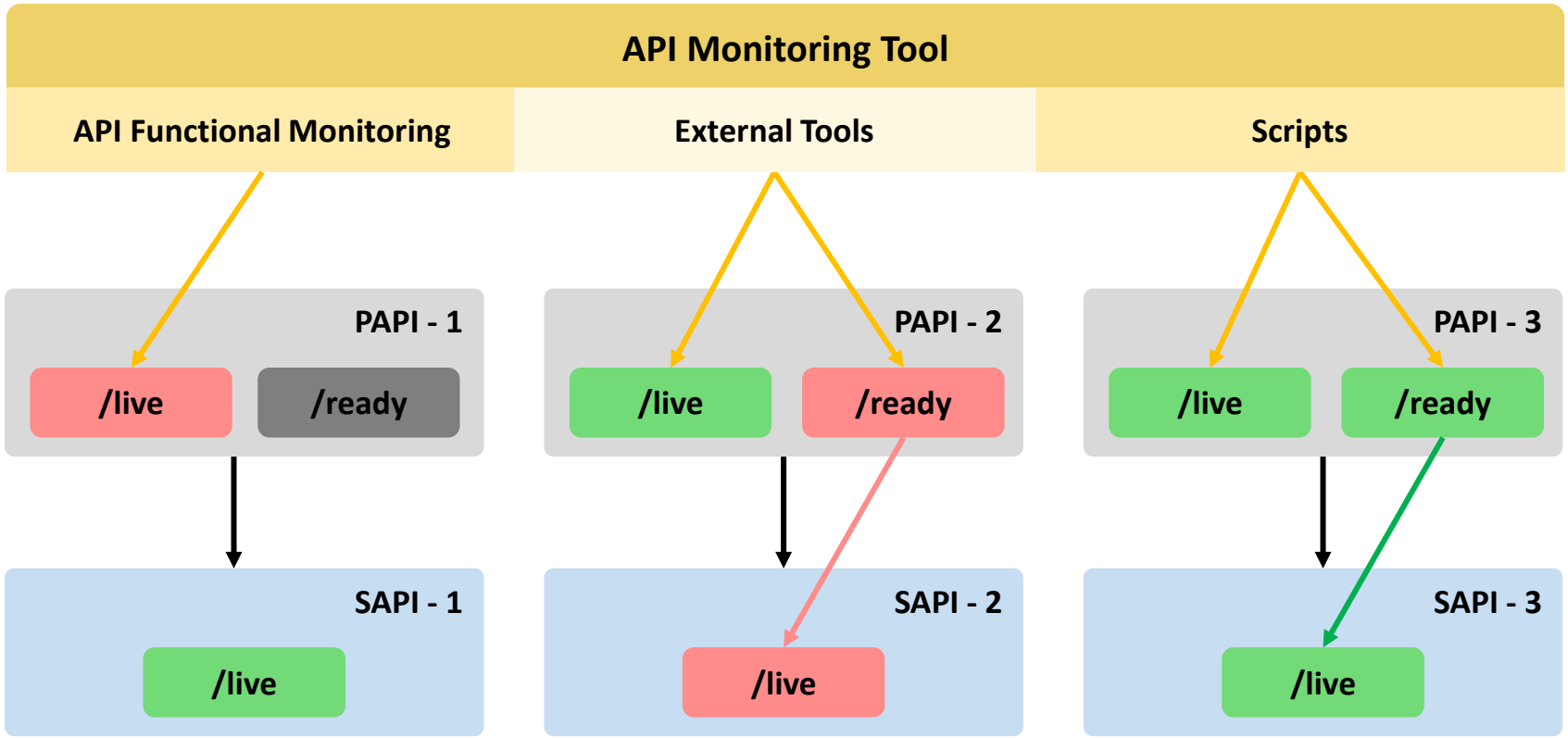
Health Endpoints

- Liveness Probe
- Readiness Probe
- API Monitoring

Health Endpoints

Liveness Probe	/live	To check that application is deployed and responding to simple HTTP calls *(if fails, **restarting** the application might help)*
Readiness Probe	/ready	To check **liveness** of downstream dependencies to route real traffic *(if fails, **restarting** the application **will not** help solve the issue)*

Application	Restart helps?	Serve Traffic?
PAPI - 1	Yes	No
PAPI - 2	No	No
PAPI - 3	n.a	Yes



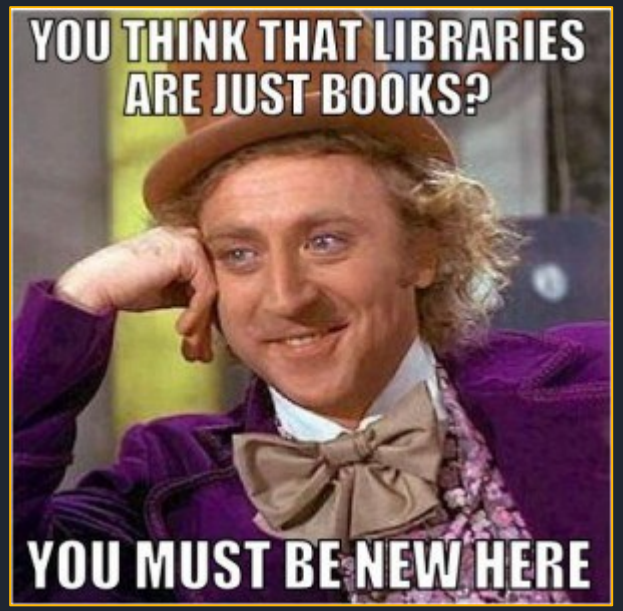
API Monitoring Test	API Endpoint to test	Optional Request Headers	Verify assertions like StatusCode, Headers, Responses	Reports to Slack, EMail, HipChat

4.1 Health Endpoints

- Liveness Probe
- Readiness Probe
- HTTP Status Code Validator

Shared Libraries

- Common Mule Artefacts/Logic
- Naming convention
- Local installation



Share Mule Artefacts as Library

Mule artefacts can be shared and reused across projects

- Common flows/sub-flows
- Common error handling
- Common health checks
- Dataweave Scripts

Mule app - common-app-logic

```
...
<groupId>com.super.org</groupId>
<artifactId>common-app-logic</artifactId>
<version>1.0.0</version>
<packaging>mule-application</packaging>
...
<plugin>
    <groupId>org.mule.tools.maven</groupId>
    <artifactId>mule-maven-plugin</artifactId>
    <version>3.3.9</version>
    <configuration>
        <classifier>mule-plugin</classifier>
    </configuration>
</plugin>
...
<dependency>
    <groupId>org.mule.connectors</groupId>
    <artifactId>mule-http-connector</artifactId>
    <version>1.6.0</version>
    <classifier>mule-plugin</classifier>
    <scope>provided</scope>
</dependency>
...
```

mvn clean install *(installs to local m2 repository)*

Mule EAPI/PAPI/SAPI

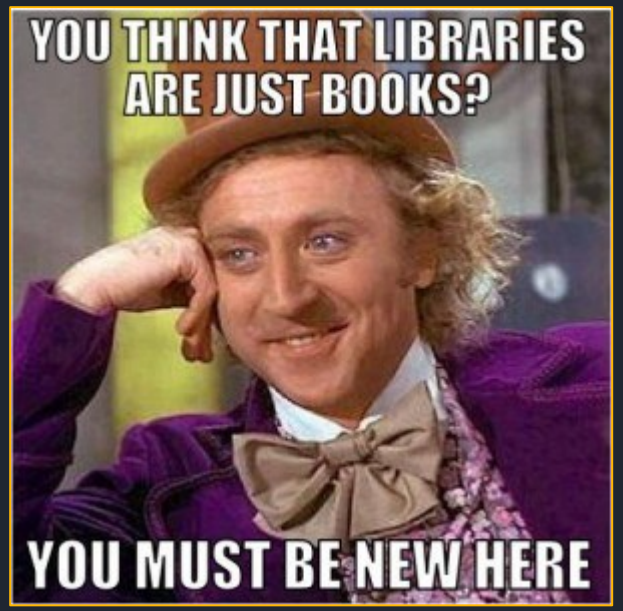
```
<dependency>
    <groupId>org.mule.connectors</groupId>
    <artifactId>mule-http-connector</artifactId>
    <version>1.6.0</version>
    <classifier>mule-plugin</classifier>
</dependency>

<dependency>
    <groupId>com.super.org</groupId>
    <artifactId>common-app-logic</artifactId>
    <version>1.0.0</version>
    <classifier>mule-plugin</classifier>
</dependency>
```

mvn clean verify -U

```
<import
    doc:name="Import"
    doc:id="84410e8b-f4fc-4ab6-abb5-4cd165afe4cf"
    file="health-common.xml" />
```

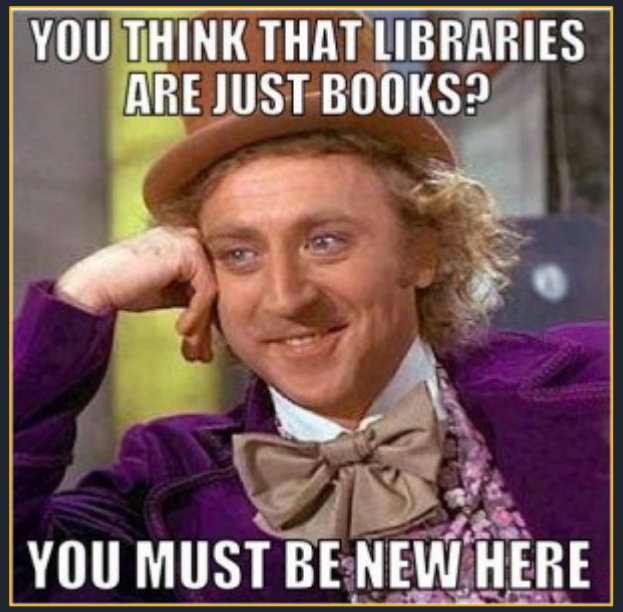
4.2 Shared Libraries

- Create a new app
- Add common mule flows
- Install to local maven repo



4.3 Using Shared Library

- Configure it in a Mule app
- Test



Custom Connector Development

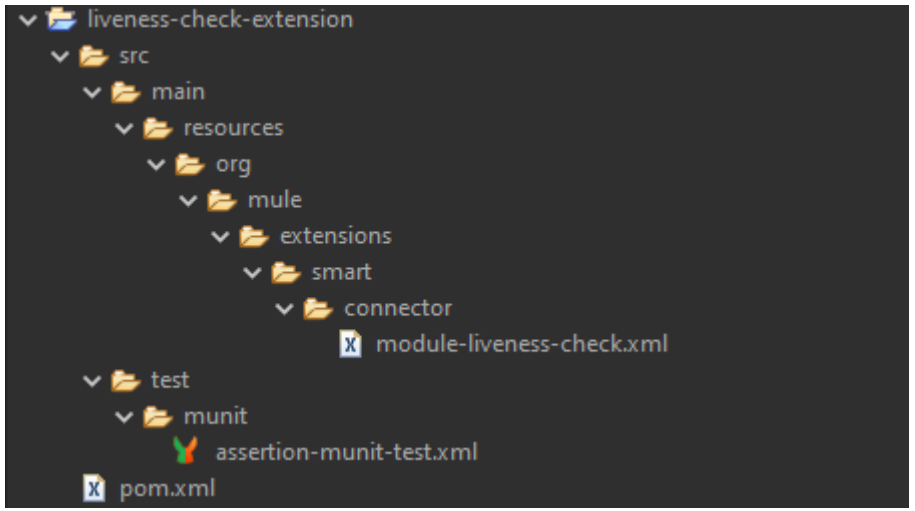
- XML SDK
- Generating the skeleton
- Project Structure
- Build & Install to Maven repo

XML SDK – Custom Mule Connector

Creating Custom Connector using Mule can be done using JAVA SDK and XML SDK

Creates a mule-plugin artefact | Mule Maven Plugin use the Exchange Mule Maven Plugin for deployment to Exchange *(mvn deploy)*

```
mvn archetype:generate \
 -DarchetypeGroupId=org.mule.extensions \
 -DarchetypeArtifactId=mule-extensions-xml-archetype \
 -DarchetypeVersion=1.2.0 \
 -DgroupId=${org_id} \
 -DartifactId=liveness-check-extension \
 -DmuleConnectorName=liveness-check
```




```xml
<module>
    <operation>

        <parameters>
            <parameter ...
        </parameters>

        <body> ...
        </body>

        <output ...

        <errors>
            <error type="" />
        </errors>
    </operation>
</module>
```

module-liveness-check.xml

```xml
<module>
    <operation>

        <parameters>
            <parameter
                name="url"
                displayName="URL"
                type="string"
                use="REQUIRED" />
        </parameters>

        <body>
            <http:request
                config-ref="livenessHttpRequestConfig"
                method="GET"
                url="#[vars.url]">
                <mule:error-mapping
                    sourceType="HTTP:UNAUTHORIZED"
                    targetType="LIVENESS-CHECK:UNAUTHORIZED" />
            </http:request>
            <mule:set-payload value="#[true]" />
        </body>

        <output
            type="boolean" />

        <errors>
            <error type="UNAUTHORIZED" />
        </errors>
    </operation>
</module>
```

mvn clean install *(installs to local m2 repository)*

mvn clean deploy *(deploy to exchange – requires additional config)*

XML SDK – Catalog

Standard Data Types for <property> and <parameter> are primitive – *boolean, number, date, datetime, time, localdatetime, string, timezone, binary, any, regex*

For Data Types with **complex** structure we can create a **CATALOG** of data type and inject them into the **MODULE**

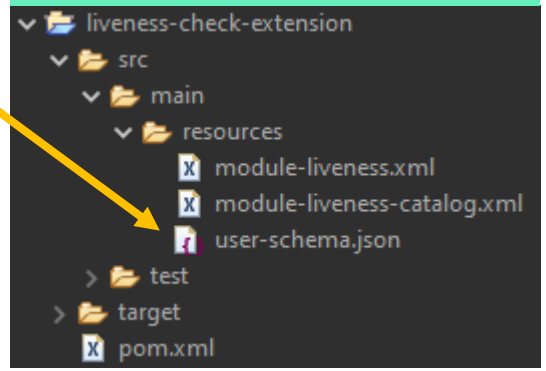
module-liveness-catalog.xml

```
38  <catalogs xmlns="http://www.mulesoft.org/schema/mule/types">
39      <catalog
40          name="UserJsonType"
41          format="application/json">
42          <schema
43              format="application/json+schema"
44              location="./user-schema.json" />
45      </catalog>
46  </catalogs>
```

user-schema.json

```
1  {
2    "type": "object",
3    "properties": {
4      "age": {
5        "type": "integer"
6      },
7      "name": {
8        "type": "string"
9      }
10   },
11   "additionalProperties": false
12 }
```

Project File Structure



module-liveness.xml

```
31  <module>
32      <operation>
33          <parameters>
34              <parameter
35                  name="name"
36                  displayName="Name"
37                  type="string"
38                  use="REQUIRED" />
39              <parameter
40                  name="age"
41                  displayName="Age"
42                  type="integer" />
43          </parameters>
44
45          <body>
46              <ee:transform>
47                  <ee:set-payload><![CDATA[
48              %dw 2.0
49              %output application/json encoding='UTF-8'
50              ---
51              {
52                "name" : upper(vars.name),
53                "age" : vars.age as Number
54              }
55              ]]></ee:set-payload>
56                </ee:transform>
57          </body>
58          <output type="UserJsonType" />
59      </operation>
60  </module>
```

4.4 Custom Connector Development

- XML SDK
- Generating the skeleton
- Make few changes

4.5 Custom Connector Development

- XML SDK
  - Add business logic
  - Add error handling
  - Install to local Maven repo

4.6 Custom Connector Development

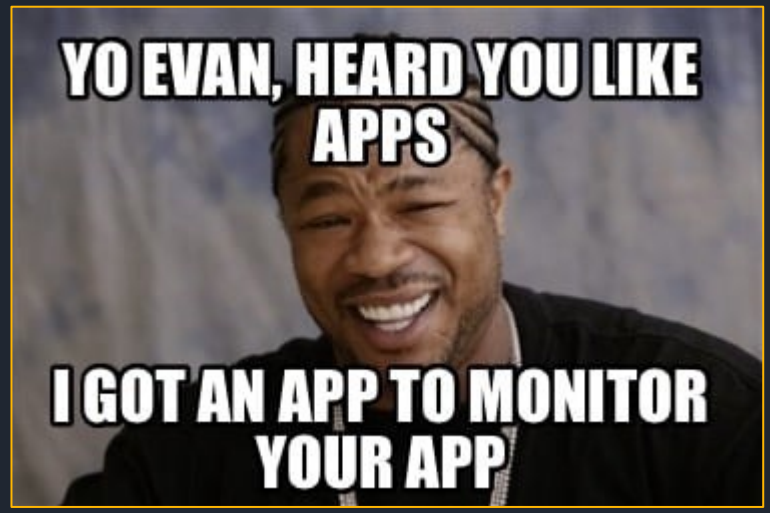
- Add the new custom connector as a Dependency
- Use the new operation

4.7 Custom Connector Development

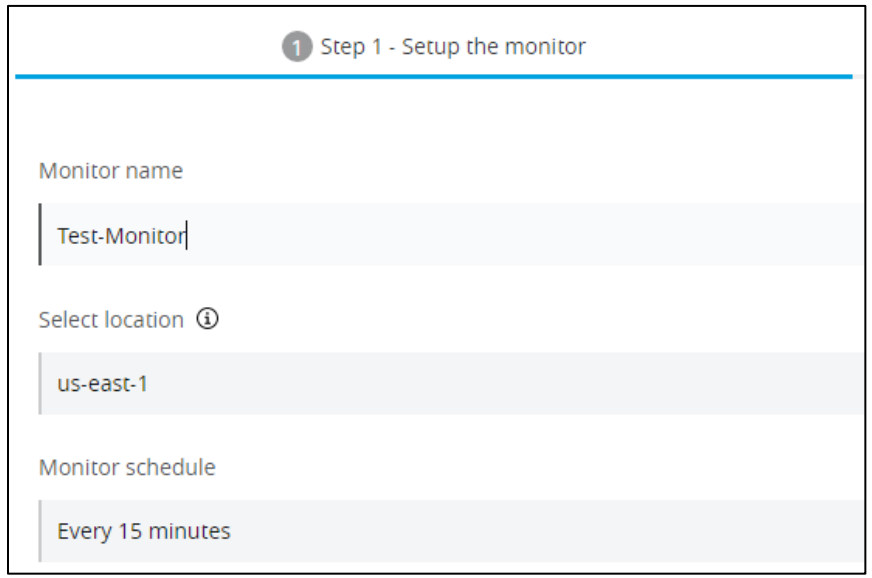
- Check Custom Error Mapping
- Talk about Catalog
- Deploy it to Exchange

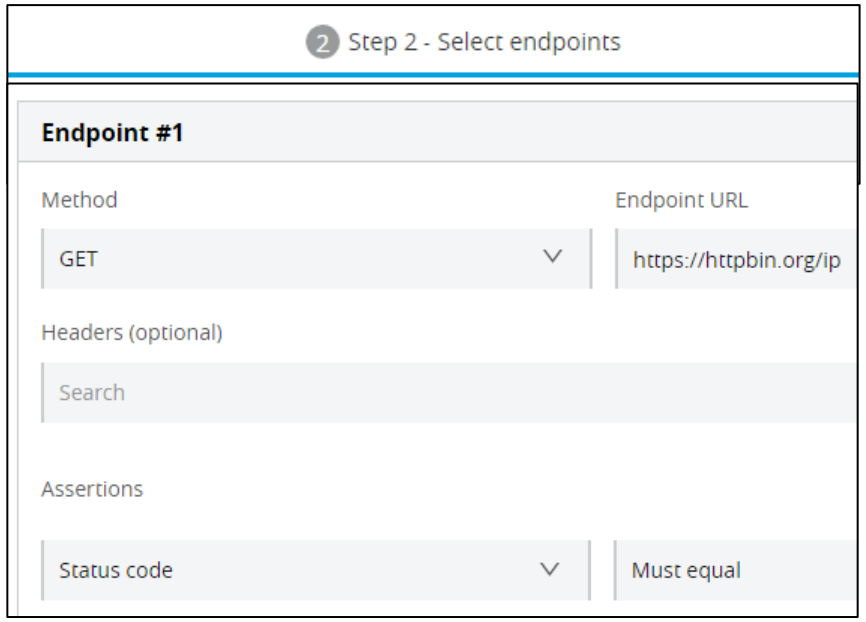
Monitoring

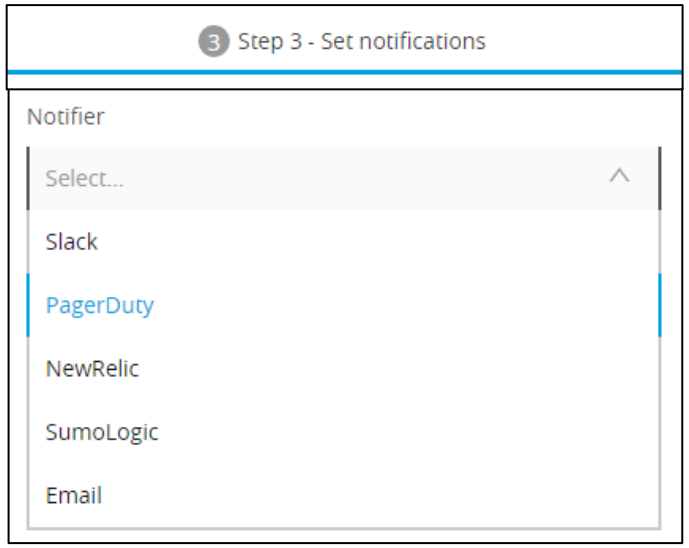
- Anypoint Functional Monitoring Steps

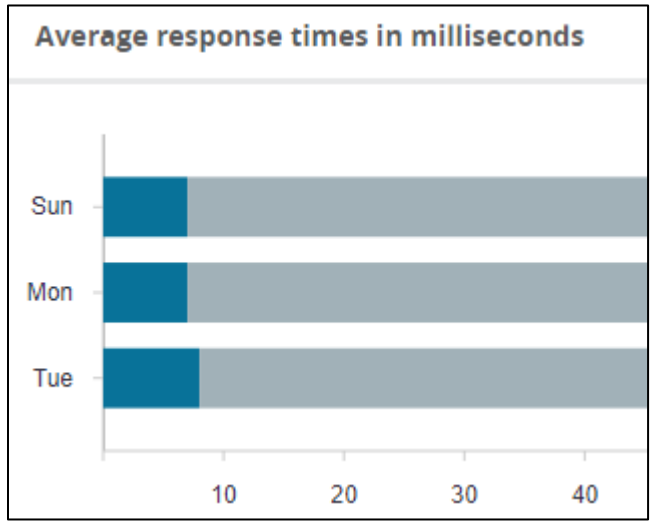


Anypoint Functional Monitoring









Create Monitor | Customize Monitor | Download Monitor | Upload Monitor

4.8 Monitoring

- Monitoring using an External system
- Validate Status Code
- Schedule it every 1 minute
- Send Failure reports

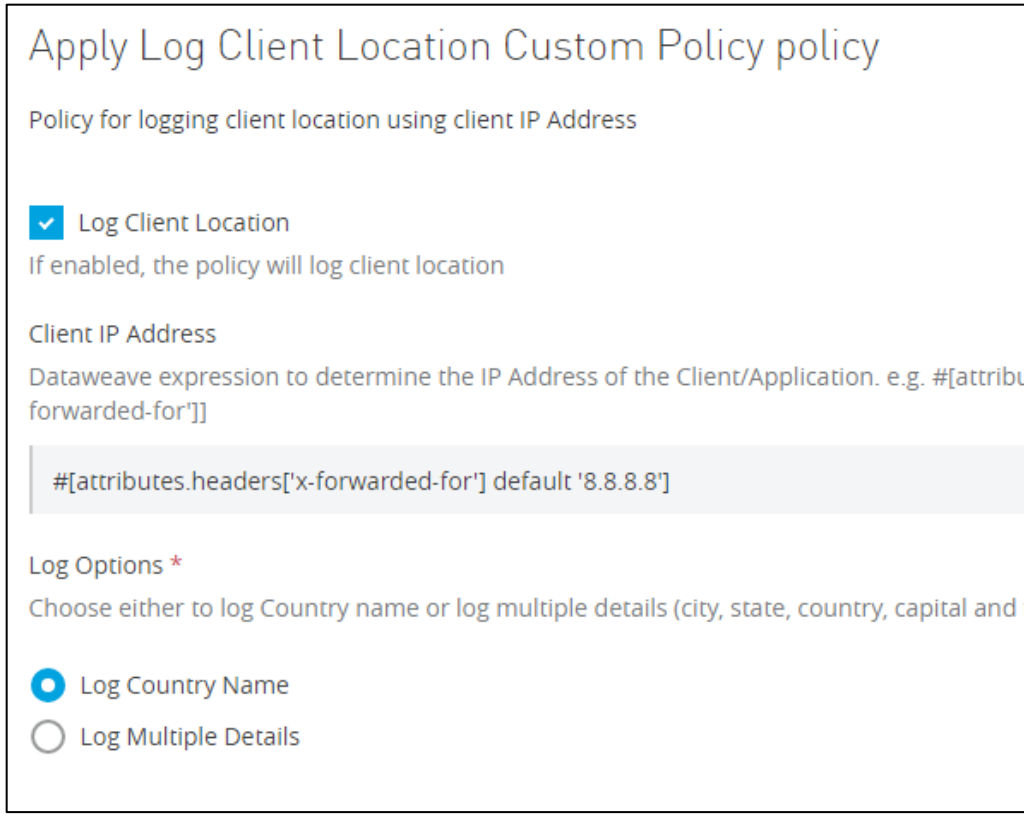
Section 4 - Completed

#4	Mule Application Health Checks and Custom Connector Development

- Configure health endpoint
- Understand Liveness vs Readiness concepts
- Create a new library with common code
- Install it to local maven repo
- Create a new connector using XML SDK
- Build, Package and Publish it to Exchange
- API Monitoring

#5 Custom API Policy Development, Offline Policies Management

- Generate Custom API Policy template
- Build the policy
- Package and deploy to Exchange
- Apply policy using API Manager
- Apply policy Offline
- Use Handle Bar to create complex policies



Custom API Policy

- What/Why/How?
- Project Structure
- Development options
- Handlebars

Custom API Policy - Development

Custom Policies can extending existing functionality or defining new ones.

Policies can be applied/deployed locally as offline policies. | For applying online, policies should be deployed to Exchange

```
mvn -Parchetype-repository archetype:generate \
-DarchetypeGroupId=org.mule.tools \
-DarchetypeArtifactId=api-gateway-custom-policy-archetype \
-DarchetypeVersion=1.2.0 \
-DgroupId=${orgId} \
-DartifactId=${policyName} \
-Dversion=1.0.0 \
-Dpackage=mule-policy
```

```
test-policy
  src
    main
      mule
        template.xml
  mule-artifact.json
  pom.xml
  test-policy.yaml
```

For applying policy using API Manager

mvn deploy *(deploys to Anypoint Exchange)*

template.xml

```
...
<http-policy:proxy name="{{{policyId}}}-policy">
        <http-policy:source>

            <set-variable
                variableName="msg"
                value="{{{message}}}" />

            <logger
                message="#[vars.msg]" />

            <http-policy:execute-next/>

        </http-policy:source>
</http-policy:proxy>
```

test-policy.yaml

```
id: custom-log-message
name: Custom Log Message
description: Policy for logging
category: Custom
type: custom
resourceLevelSupported: true
encryptionSupported: false
standalone: true
requiredCharacteristics: []
providedCharacteristics: []
configuration:
  - propertyName: message
    name: Enter Message
    description: |
      Enter value to Log
    type: string
    optional: true
    sensitive: false
```

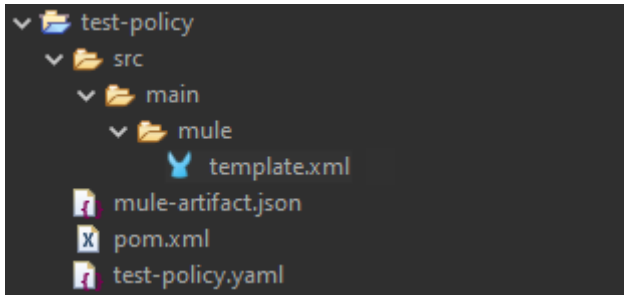
pom.xml

```
...
    <groupId>1a1a1a1a-2b2b2b2b-3c3c3c3c</groupId>
    <artifactId>test-policy</artifactId>
    <version>1.0.0-SNAPSHOT</version>

    <name>test-policy</name>
    <packaging>mule-policy</packaging>
...
    <build>
        <plugins>
            <plugin>
                <groupId>org.mule.tools.maven</groupId>
                <artifactId>mule-maven-plugin</artifactId>
                <version>${mule.maven.plugin.version}</version>
                <extensions>true</extensions>
            </plugin>
            <plugin>
                <groupId>org.apache.maven.plugins</groupId>
                <artifactId>maven-deploy-plugin</artifactId>
                <executions> ...
                </executions>
            </plugin>
        </plugins>
    </build>

    <distributionManagement>
        <repository>
            <id>exchange-server</id>
            <name>Corporate Repository</name>
            <url>${exchange.url}</url>
            <layout>default</layout>
        </repository>
    </distributionManagement>
...
```

Custom API Policy – Handlebars

Handlebar is a templating framework and can access config values from YAML file

Using **if conditions**, it decides which section of policy is executed based on user input



template.xml

```
...
<http-policy:proxy name="{{{policyId}}}-policy">
        <http-policy:source>

            <set-variable
                variableName="msg"
                value="{{{message}}}" />

            <logger
                message="#[vars.msg]" />

           <http-policy:execute-next/>

            {{#if upperCase}}
            <logger
                message="#[upper(vars.msg)]" />
            {{/if}}

        </http-policy:source>
</http-policy:proxy>
...
```

test-policy.yaml

```
id: custom-log-message
name: Custom Log Message
description: Policy for logging
category: Custom
type: custom
resourceLevelSupported: true
encryptionSupported: false
standalone: true
requiredCharacteristics: []
providedCharacteristics: []
configuration:
  - propertyName: message
    name: Enter Message
    description: Enter value to Log
    type: string
    optional: true
    sensitive: false

  - propertyName: upperCase
    name: Enable Upper Case
    description: Logs message in upper case.
    type: boolean
    optional: true
    defaultValue: false
```

Custom API Policy

MCD2

Simple Custom Policy – Straight Forward Logic

Apply Log Client Location Custom Policy policy

Policy for logging client location using client IP Address

☑ Log Client Location

If enabled, the policy will log client location

Client IP Address

Dataweave expression to determine the IP Address of the Client/Application. e.g. #[attributes.headers['x-forwarded-for']]

#[attributes.headers['x-forwarded-for']]

Choose runtime range

◉ Apply to all supported runtimes (4.1.1 and above)

○ Apply to a specific set of runtimes

Cancel

#
Request Originated from **HYDERABAD** city, within **TELANGANA** state of **INDIA** (**New Delhi**) in **ASIA/KOLKATA** timezone.
#

Complex Custom Policy – Choice, IF Logic, DependsOn Conditions

Apply Log Client Location Custom Policy policy

Policy for logging client location using client IP Address

☑ Log Client Location

If enabled, the policy will log client location

Client IP Address

Dataweave expression to determine the IP Address of the Client/Application. e.g. #[attributes.headers['x-forwarded-for']]

#[attributes.headers['x-forwarded-for'] default '8.8.8.8']

Log Options *

Choose either to log Country name or log multiple details (city, state, country, capital and timezone data)

◉ Log Country Name

○ Log Multiple Details

###*
Request Originated Country Name – **INDIA**
###*

5.1 Custom API Policy

- Generate skeleton
- POM Modifications

5.2 Custom API Policy - Development

- Understand Client IP Logging
- Add Business Logic to Policy
- Add UI Properties
- Package the policy

5.3 Custom API Policy - Deployment

- Deploy to Exchange
- Configure in API Manager

Custom API Policy - Offline

- Pre-requisites
- Applying Offline Custom Policy
- Removing Offline Custom Policy

Managing Offline Custom Policy – Apply & Remove

Prerequisites Before applying an offline custom policy	API Gateway Capabilities enabled in Mule Runtime Engine
	API configured with a basic or proxy endpoint exists in API Manager
	Mule application is linked with API Autodiscovery & deployed with a HTTP(s) flow
Applying Offline Custom Policy	**mvn clean package** >>>> Copy the **JAR** >>>> ***MULE_HOME_DIR/policies/policy-templates***
	Create JSON file >>>> ***MULE_HOME_DIR/policies/offline-policies***
Removing Offline Custom Policies	Delete JSON file >>>> ***MULE_HOME_DIR/policies/offline-policies***

log-client-location-custom-policy-definition.json

```json
{
  "template" : {
    "groupId" : "c738e3c8-08cf-462d-bb9a-1553ebf5008e",
    "assetId" : "log-client-location-custom-policy",
    "version" : "1.1.0"
  },
  "api": [
    {
      "id": "17719055"
    }
  ],
  "order": 1,
  "configuration" : {
    "ipAddress" : "1.1.1.1"
  }
}
```

policy-definition.json

```json
{
  "id" : "log-client-location-custom-policy-definition",
  "template" : {
    "groupId" : "c738e3c8-08cf-462d-bb9a-1553ebf5008e",
    "assetId" : "log-client-location-custom-policy",
    "version" : "1.1.0"
  },
  "api" : [ {
    "id" : 17719055
  } ],
  "online" : false,
  "order" : 1,
  "configuration" : {
    "ipAddress" : "1.1.1.1"
  }
}
```

5.4 Custom API Policy - Offline

- Copy artefacts to Mule_Home Directories
- Deploy App
- Test
- Remove Policy

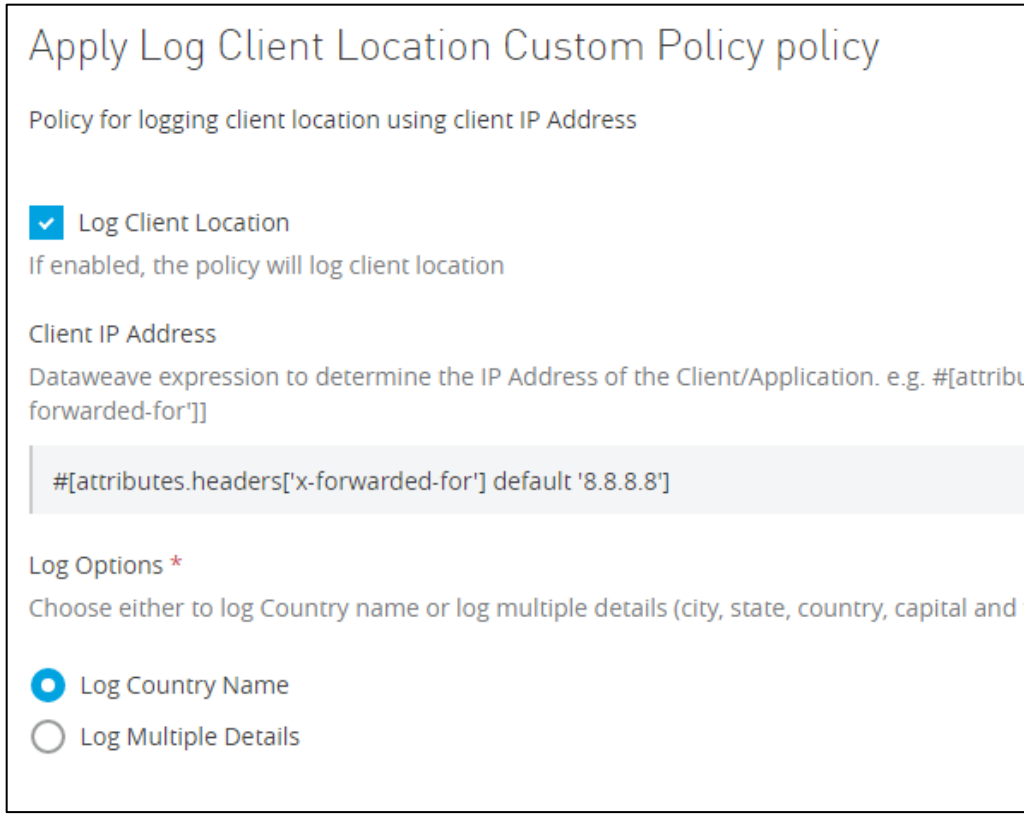
5.5 Custom API Policy - Complex

- Add handlebar logic
- Package the policy
- Deploy to Exchange
- Apply in API Manager
- Check in CloudHub Logs

Section 5 - Completed

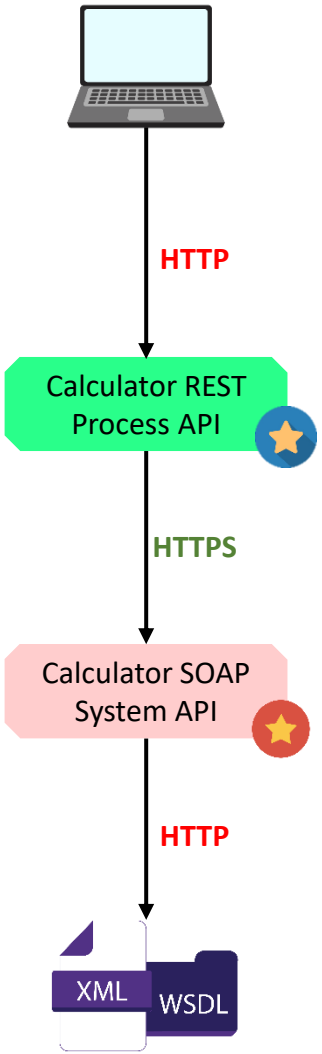
#5 Custom API Policy Development, Offline Policies Management

- Generate Custom API Policy template
- Build the policy
- Package and deploy to Exchange
- Apply policy using API Manager
- Apply policy Offline
- Use Handle Bar to create complex policies



#6 Consuming SOAP Web Services using TLS Certificates (one-way and two-way)

- Consume SOAP Service over plain HTTP Protocol
- Consume SOAP Service using one-way-tls
- Consume SOAP Service using two-way-tls
- Configure TLS Certs for a SOAP WebConsumer connector
- Deploy application to Mule 4 Standalone server



SOAP WebService

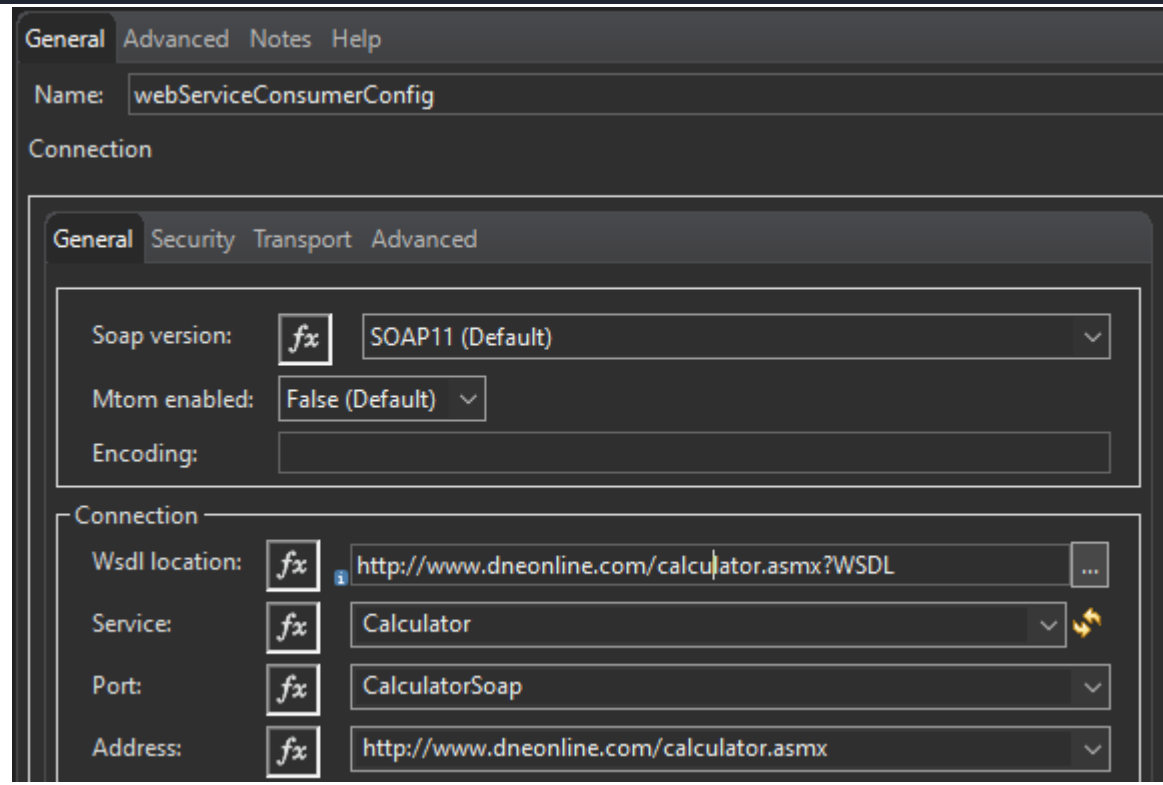
- Add Module
- Configure TLS Certs

Consume SOAP Webservice with TLS certificates

```xml
<dependency>
    <groupId>org.mule.connectors</groupId>
    <artifactId>mule-wsc-connector</artifactId>
    <version>1.6.7</version>
    <classifier>mule-plugin</classifier>
</dependency>
```

```xml
<wsc:config
        name="webServiceConsumerConfig">
    <wsc:connection
            wsdlLocation="http://www.dneonline.com/calculator.asmx?WSDL"
            service="Calculator"
            port="CalculatorSoap"
            address="http://www.dneonline.com/calculator.asmx">
        <wsc:custom-transport-configuration>
            <wsc:http-transport-configuration
                    requesterConfig="httpTlsSoapConsume" />
        </wsc:custom-transport-configuration>
    </wsc:connection>
</wsc:config>
```

```xml
<http:request-config name="httpTlsSoapConsume">
    <http:request-connection
            protocol="HTTPS"
            tlsContext="tlsContext" />
</http:request-config>
```



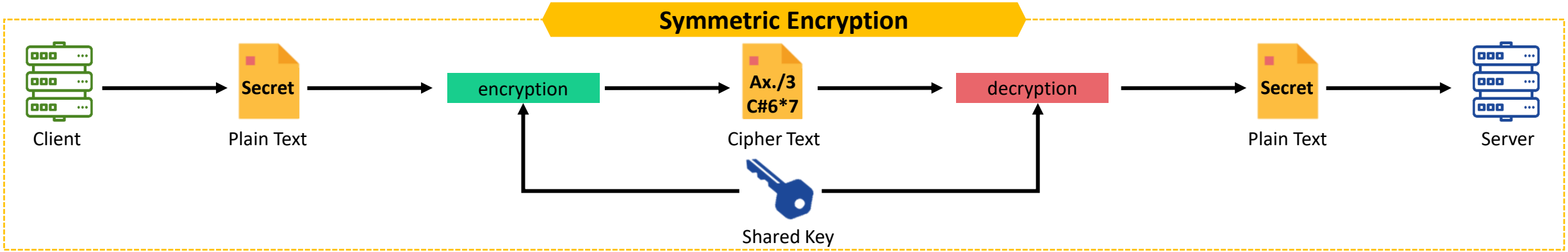

```xml
<tls:context name="tlsContext">
    <tls:trust-store
            path="server-truststore.jks"
            password="123456"
            type="jks" />
    <tls:key-store
            type="jks"
            path="server-keystore.jks"
            alias="server"
            keyPassword="123456"
            password="123456" />
</tls:context>
```

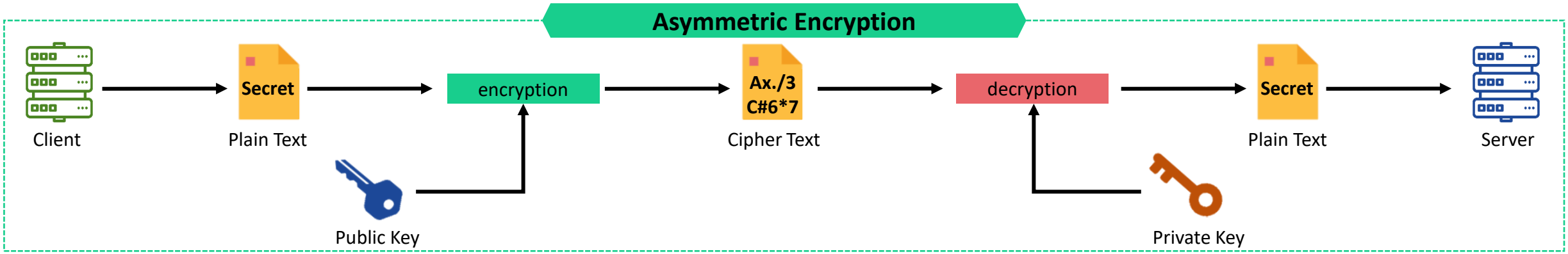
Secure Data in Transit

- Secure Communication – Using Cryptography
- Digital Certificates
- One-way TLS
- Two-way TLS
- Create Keys & Certificates (JKS) 2-Way-TLS
- Create Keys & Certificates (PKCS12) 2-Way-TLS

Secure Communication – Using Cryptography

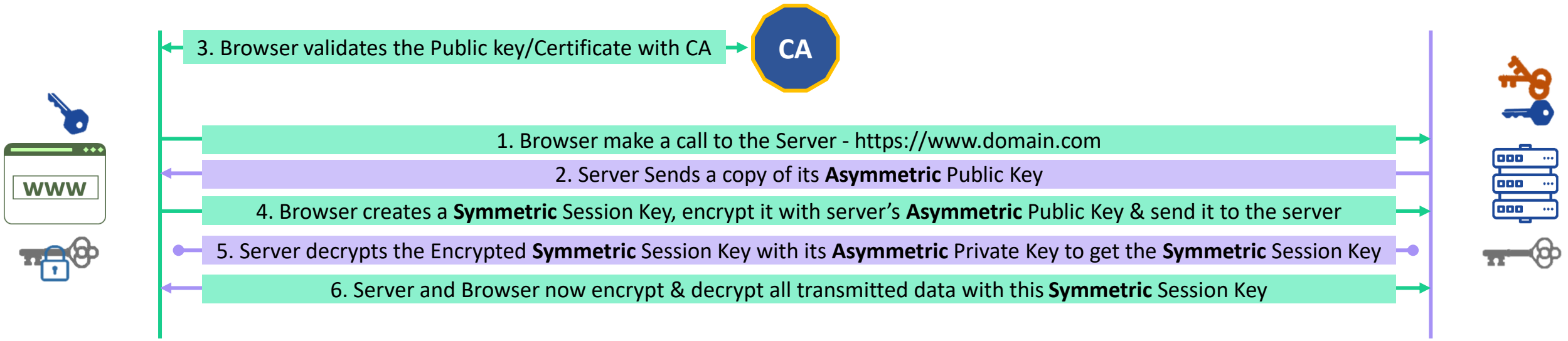
Symmetric Cryptography	Client and Server use the **same key** for encryption and decryption of messages		
	Same Key	Speed of encryption/decryption is Very **FAST**	Size of encrypted text is SAME or **LESS** than the original plaintext
Asymmetric Cryptography	Server issues a **Public Key** to the client to encrypt the messages | Server has a **Private Key** which is the **only key** to decrypt the message		
	Different Keys	Speed of encryption/decryption is **SLOWER**	Size of encrypted text is **MORE** than the original plaintext
Digital Certificates	Uses public/private key certificate signed by a **Trusted Authority (or self signed)** for communications		

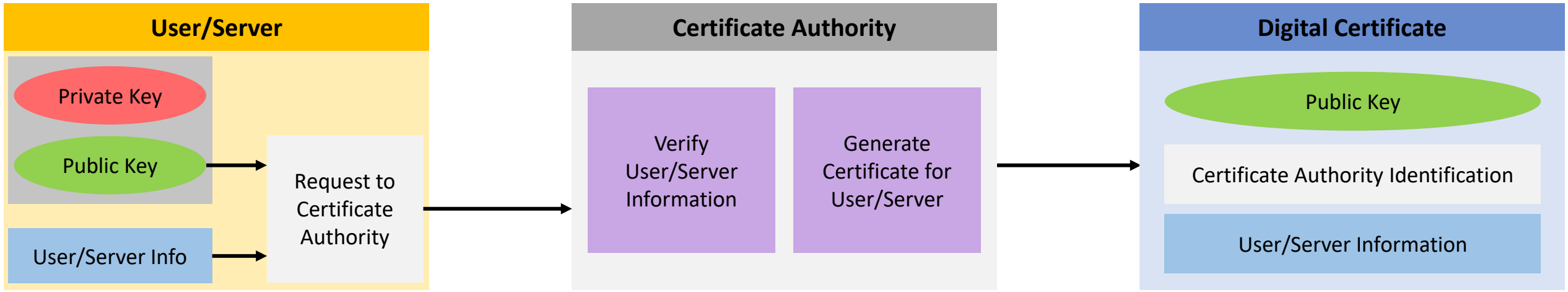




Digital Certificates

Digital Certificates	Uses public/private key certificate signed by a **Trusted Authority (or self signed)** for communications
Certificate Authority	CA is an entity that validate the identities of entities and bind them to Public Keys to issue digital certificates *(ex - GoDaddy, Symantec)*
Self-Sign Certificate	A self-signed certificate is one that is not signed by a CA *(useful in test environments)*

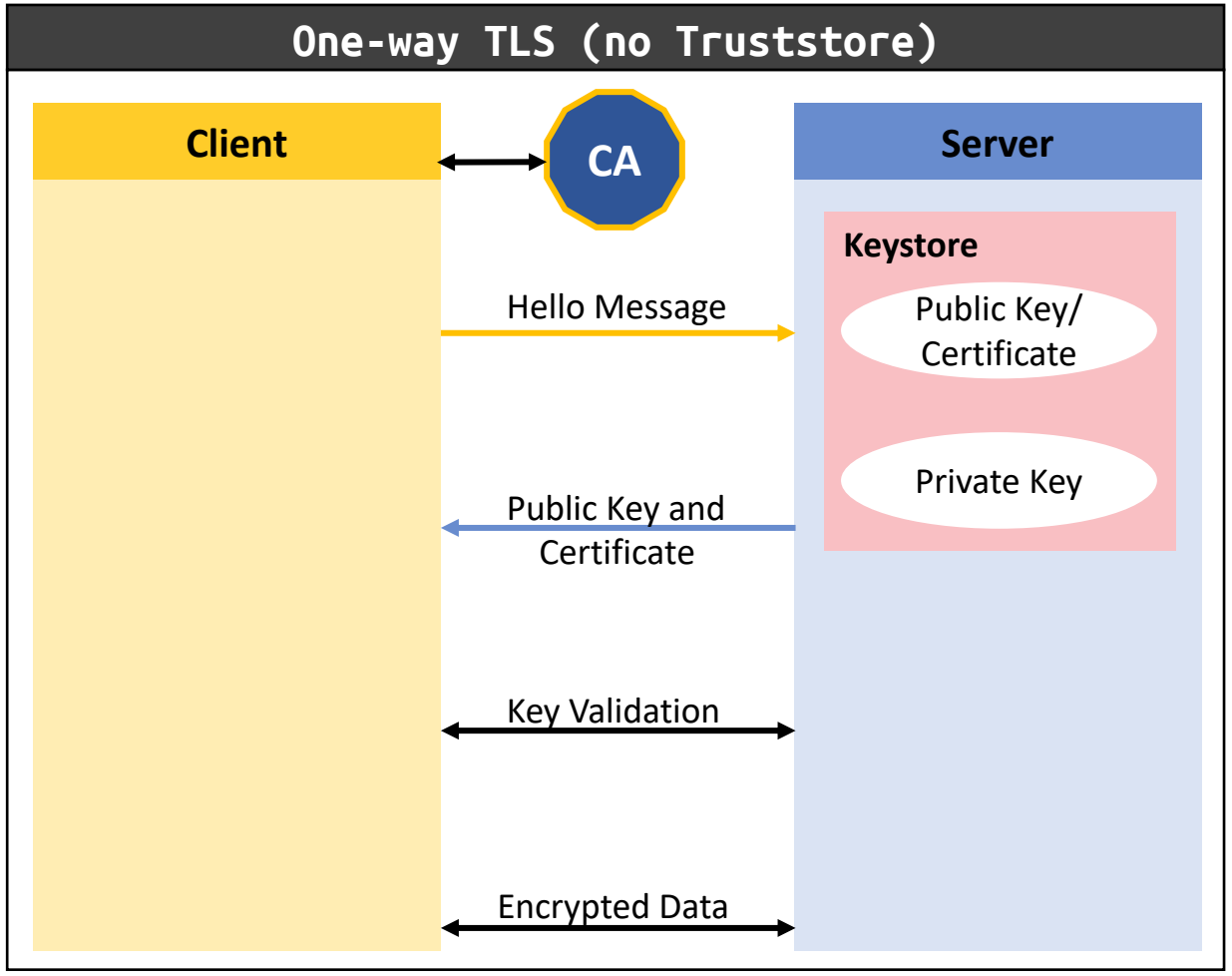


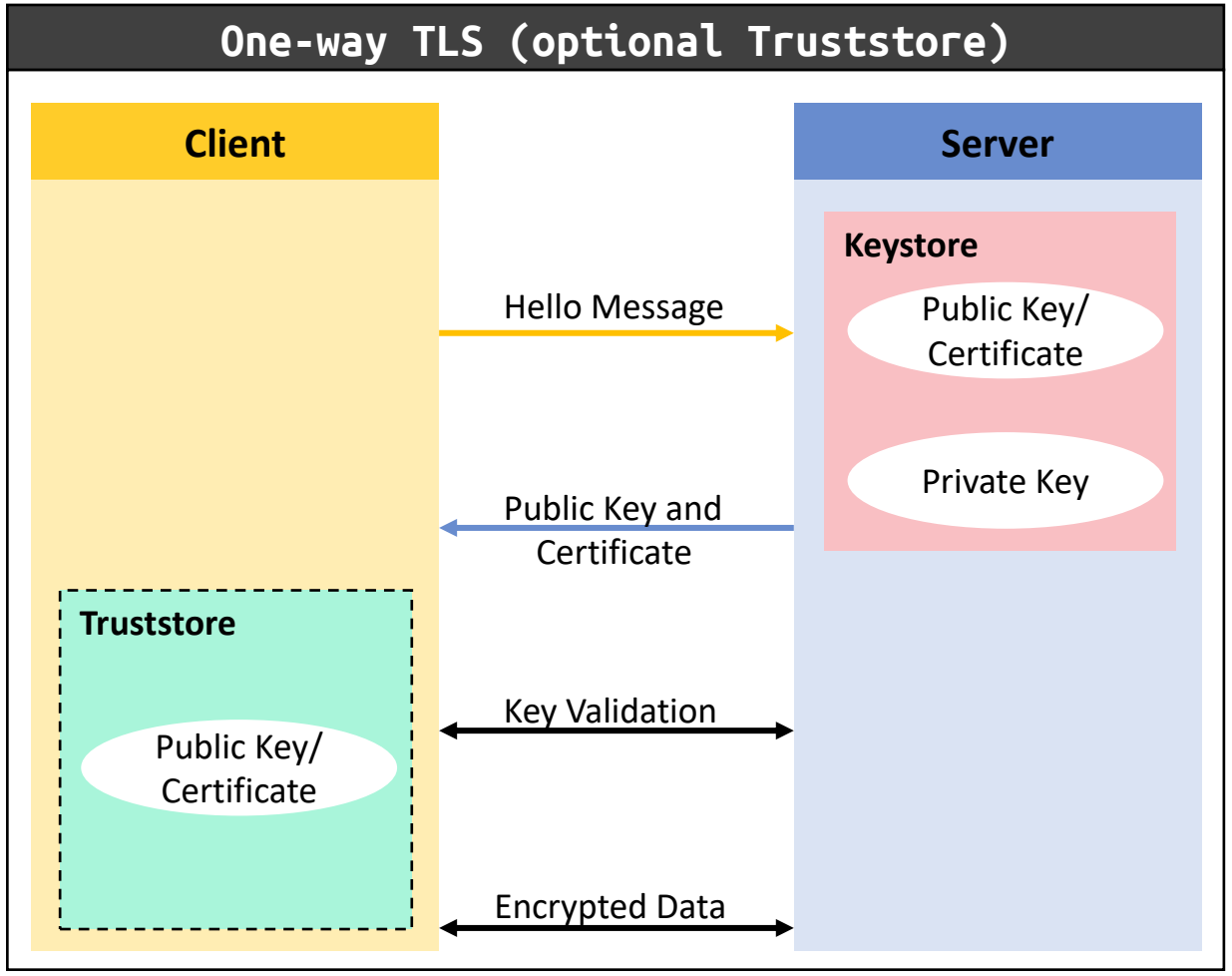


One-way TLS

Keystore	Is a **repositories** that contains **Public certificates**, plus the corresponding **Private key** for clients or servers.
Truststore	Contains trusted certificates on a TLS client used to **validate** a TLS Server's Public certificate presented to the client *(typically self-signed certificates)*

In **One-way TLS** the **client** always **verifies** the **server** certificates and the **server** **NEVER** verifies the **client** certificates

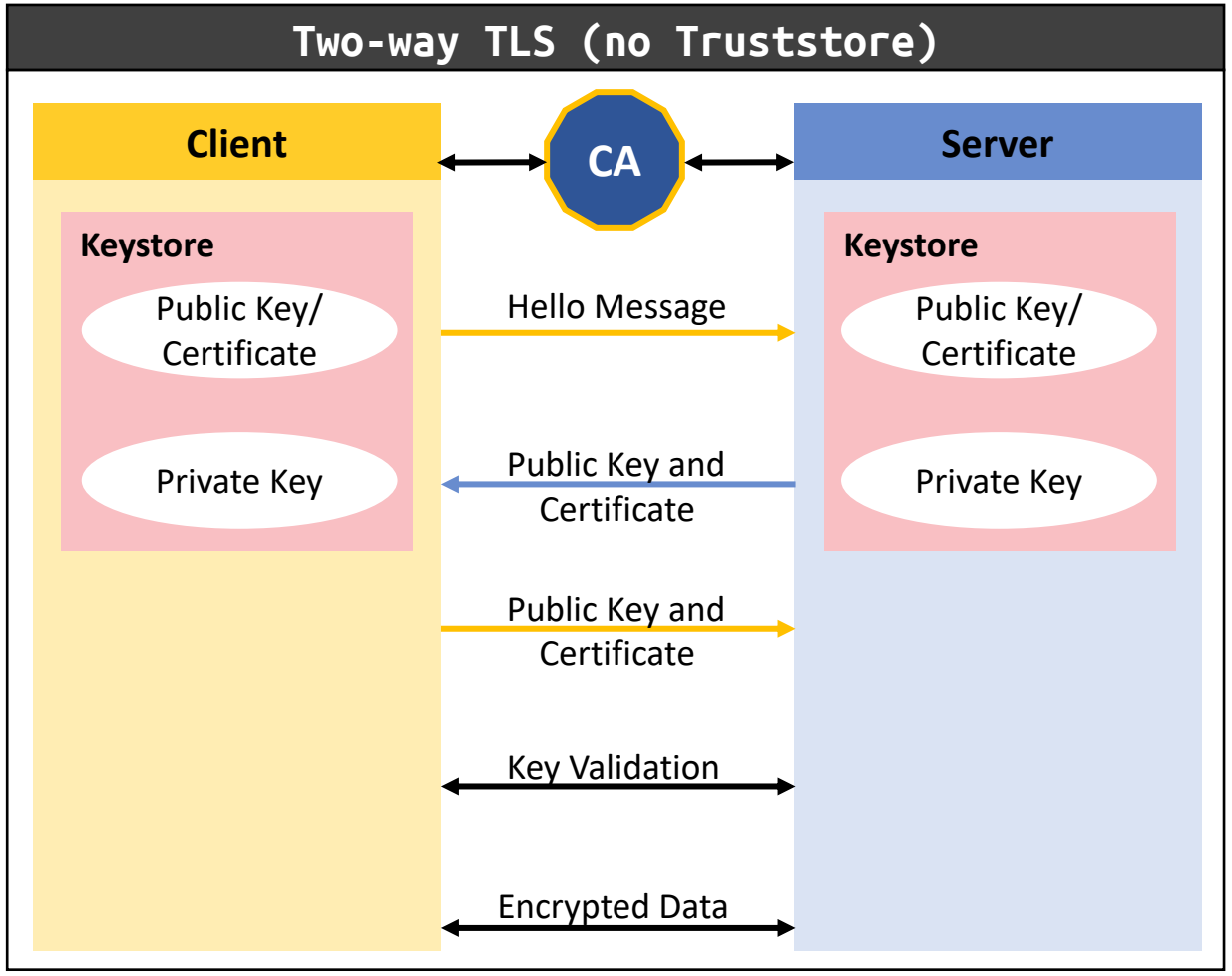


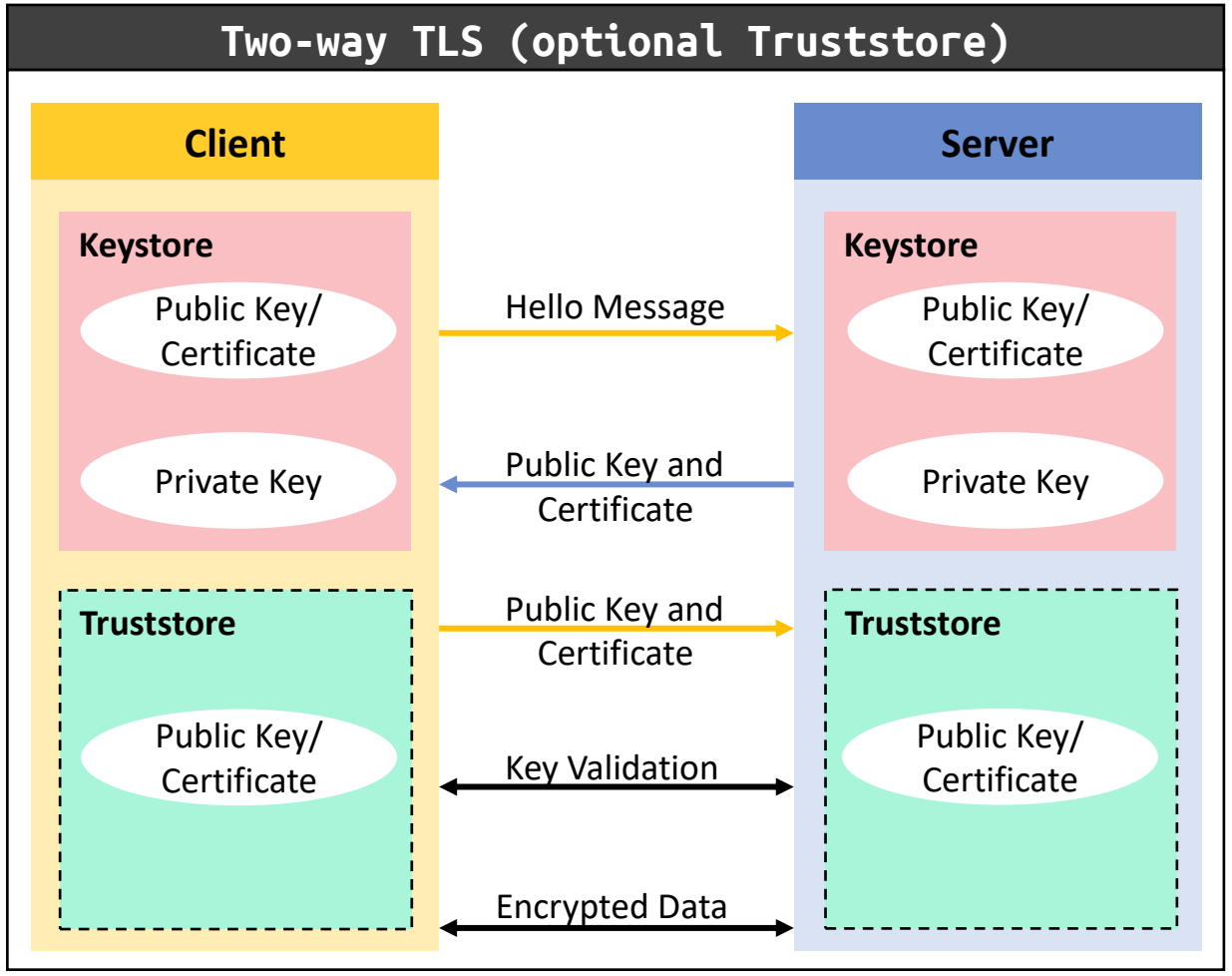


Two-way TLS

Keystore	Is a **repositories** that contains **Public certificates**, plus the corresponding **Private key** for clients or servers.
Truststore	Contains trusted certificates on a TLS client used to **validate** a TLS Server's Public certificate presented to the client *(typically self-signed certificates)*

In **Two-way TLS** the **client** verifies the **server** certificates and server **verifies** the client certificates.





Create Keys & Certificates (JKS) 2-Way-TLS

Create A Private/Public Key Pair - ***server-keystore.jks*** file contains the newly generated public and private key pair

```
keytool -genkey -alias mule-server -keysize 4096 -keyalg RSA -keystore server-keystore.jks
```

Extract A Self-signed Certificate From The Keystore - ***server_public.crt*** file that contains a certificate signed with the private key in the ***server_keystore.jks***

```
keytool -export -alias mule-server -keystore server-keystore.jks -file server_public.crt
```

Add A Certificate To A Truststore - certificate in ***server_public.crt*** has been added to the new truststore named ***client-truststore.jks***

```
keytool -import -alias mule-client-public -keystore client-truststore.jks -file server_public.crt
```

Server Certificates | **Client Certificates**

Create A Private/Public Key Pair - ***client-keystore.jks*** file contains the newly generated public and private key pair

```
keytool -genkey -alias mule-client -keysize 4096 -keyalg RSA -keystore client-keystore.jks
```

Extract A Self-signed Certificate From The Keystore - ***client_public.crt*** file that contains a certificate signed with the private key in the ***client_keystore.jks***

```
keytool -export -alias mule-client -keystore client-keystore.jks -file client_public.crt
```

Add A Certificate To A Truststore - certificate in ***client_public.crt*** has been added to the new truststore named ***server-truststore.jks***

```
keytool -import -alias mule-server-public -keystore server-truststore.jks -file client_public.crt
```

Create Keys & Certificates (PKCS12) 2-Way-TLS

Create A Private/Public Key Pair - ***server-keystore.p12*** file contains the newly generated public and private key pair

```
keytool -genkey -alias mule-server  -keysize 2048 -keyalg RSA -keystore server-keystore.p12 -storetype pkcs12
```

Extract A Self-signed Certificate From The Keystore - ***server_public.pem*** file that contains a certificate signed with the private key in the ***server_keystore.p12***

```
keytool -export -alias mule-server -keystore server-keystore.p12  -file server_public.pem
```

Add A Certificate To A Truststore - certificate in ***server_public.pem*** has been added to the new truststore named ***client-truststore.p12***

```
keytool -import -alias mule-client-public -keystore client-truststore.p12  -file server_public.pem
```

Server Certificates

Client Certificates

Create A Private/Public Key Pair - ***client-keystore.p12*** file contains the newly generated public and private key pair

```
keytool -genkey -alias mule-client  -keysize 2048 -keyalg RSA -keystore client-keystore.p12 -storetype pkcs12
```

Extract A Self-signed Certificate From The Keystore - ***client_public.pem*** file that contains a certificate signed with the private key in the ***client_keystore.p12***

```
keytool -export -alias mule-client -keystore client-keystore.p12  -file client_public.pem
```

Add A Certificate To A Truststore - certificate in ***client_public.pem*** has been added to the new truststore named ***server-truststore.p12***

```
keytool -import -alias mule-server-public -keystore server-truststore.p12  -file client_public.pem
```

6.1 Soap WebService – Plain HTTP

- Import SOAP **SAPI**
- Run & Test
- Deploy to Standalone Server

- Import REST **PAPI**
- Consume SOAP **SAPI**
- Test

6.2 Soap WebService > One-way TLS

- Generate server keystore and client truststore
- Package them in Mule App
- Configure keystore in SOAP **SAPI**
- Deploy to Standalone Server

- Configure client truststore in REST **PAPI**
- Deploy & Test

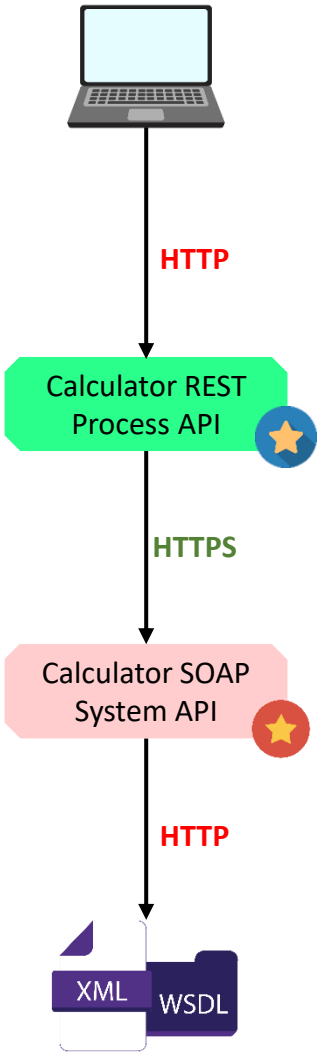
6.3 Soap WebService > Two-way TLS

- Generate client keystore and server truststore
- Package them in Mule App
- Configure keystore in REST **PAPI**
- Deploy

- Configure server truststore in SOAP **SAPI**
- Deploy to Standalone server
- Test

#6 Consuming SOAP Web Services using TLS Certificates (one-way and two-way)

- Consume SOAP Service over plain HTTP Protocol
- Consume SOAP Service using one-way-tls
- Consume SOAP Service using two-way-tls
- Configure TLS Certs for a SOAP WebConsumer connector
- Deploy application to Mule 4 Standalone server

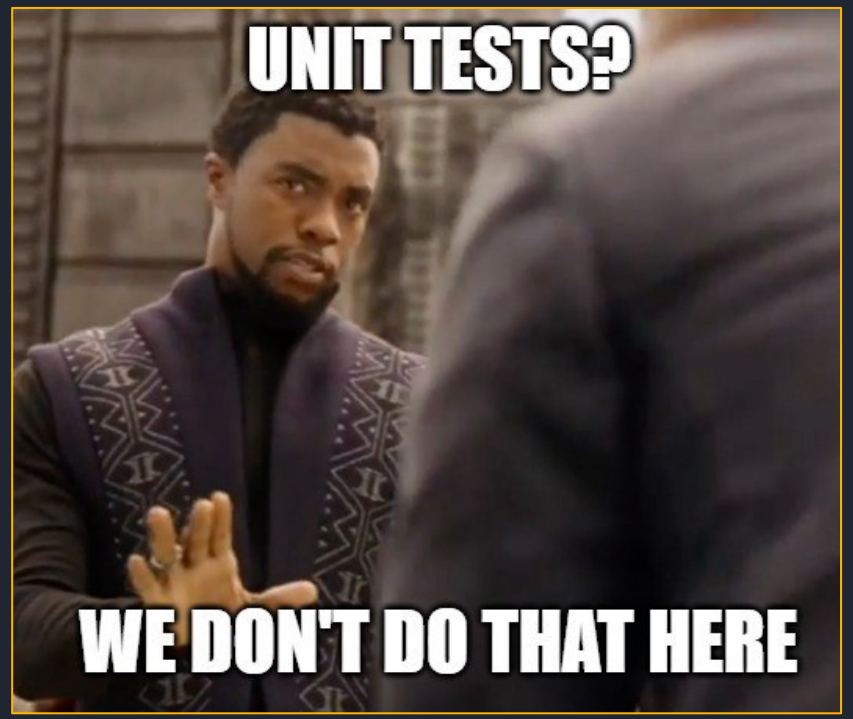


#7 MUnit Testing

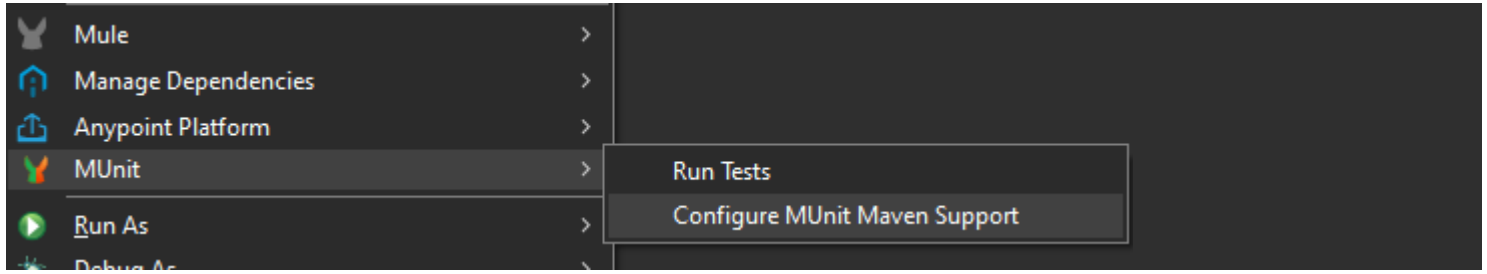
- Configure MUnit Test Suite
- Set event data
- Mock Dependencies
- Assert Responses
- Verify Calls
- Spy processors
- Refactor MUnit Code
- Test Flows with custom error handlers
- Automate MUnit using MUnit Recorder

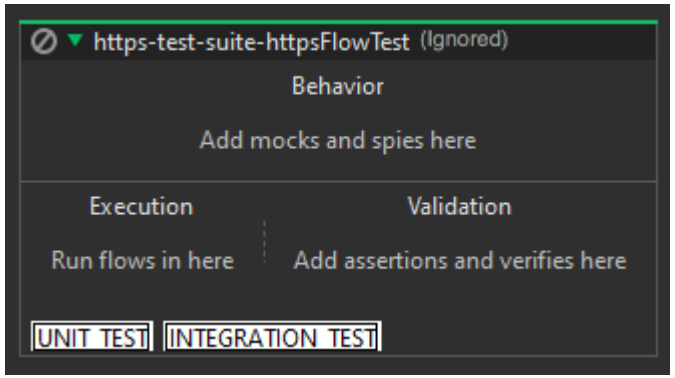
MUnit Testing

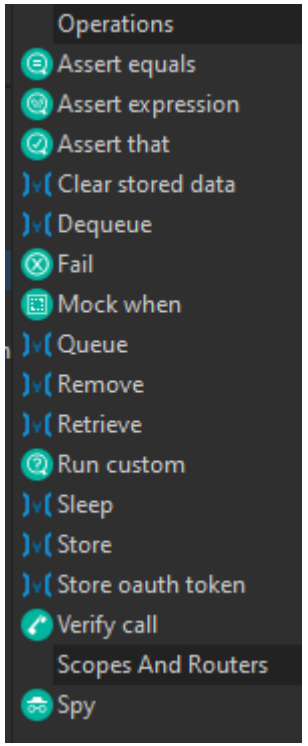
- MUnit Basics



MUnit Introduction

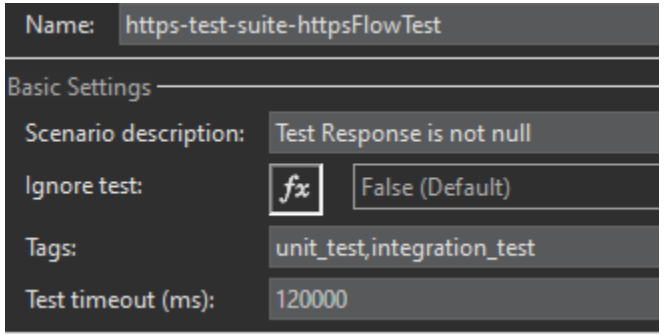


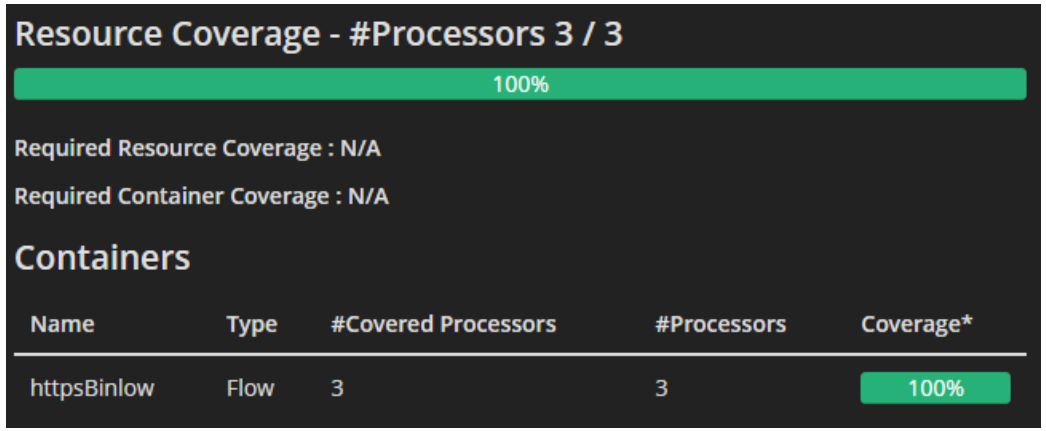





```xml
<plugin>
  <groupId>com.mulesoft.munit.tools</groupId>
  <artifactId>munit-maven-plugin</artifactId>
  <version>2.3.7</version>
  <executions>
    <execution>
      <id>test</id>
      <phase>test</phase>
      <goals>
        <goal>test</goal>
        <goal>coverage-report</goal>
      </goals>
    </execution>
  </executions>
  <configuration>
    <coverage>
      <runCoverage>true</runCoverage>
      <formats>
        <format>html</format>
      </formats>
    </coverage>
  </configuration>
</plugin>
```

```xml
<dependency>
  <groupId>com.mulesoft.munit</groupId>
  <artifactId>munit-runner</artifactId>
  <version>2.3.7</version>
  <classifier>mule-plugin</classifier>
  <scope>test</scope>
</dependency>

<dependency>
  <groupId>com.mulesoft.munit</groupId>
  <artifactId>munit-tools</artifactId>
  <version>2.3.7</version>
  <classifier>mule-plugin</classifier>
  <scope>test</scope>
</dependency>

<dependency>
  <groupId>org.mule.weave</groupId>
  <artifactId>assertions</artifactId>
  <version>1.0.2</version>
  <scope>test</scope>
</dependency>
```





MUnit Operations

Set Event

allows you to define a Mule Event

- Set Payload
- Set Attributes
- Set Errors
- Set Variables

Assert That

allows you to run assertions to validate the Mule event after the production code runs

- #[MunitTools::equalTo('example')]
- #[MunitTools::nullValue()]

Mock When

allows you to mock any processor and return a predefined result instead of executing the processor itself

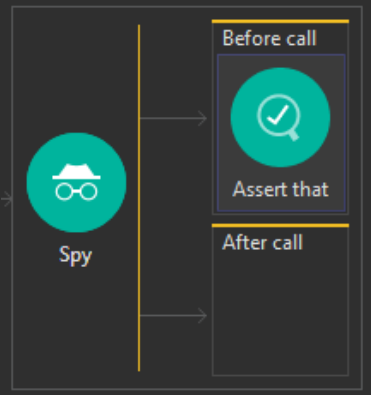
- Mocking Using Then-Return
- Mocking Using Then-Call
- Mocking Errors

Verify Call

you can validate if a specific processor has been called with a particular set of attributes a specific number of times

- times=2
- atLeast=2
- atMost=4

Spy



allows you to spy & assert what happens before and after an event processor is called

Flow Error/Exception Tests

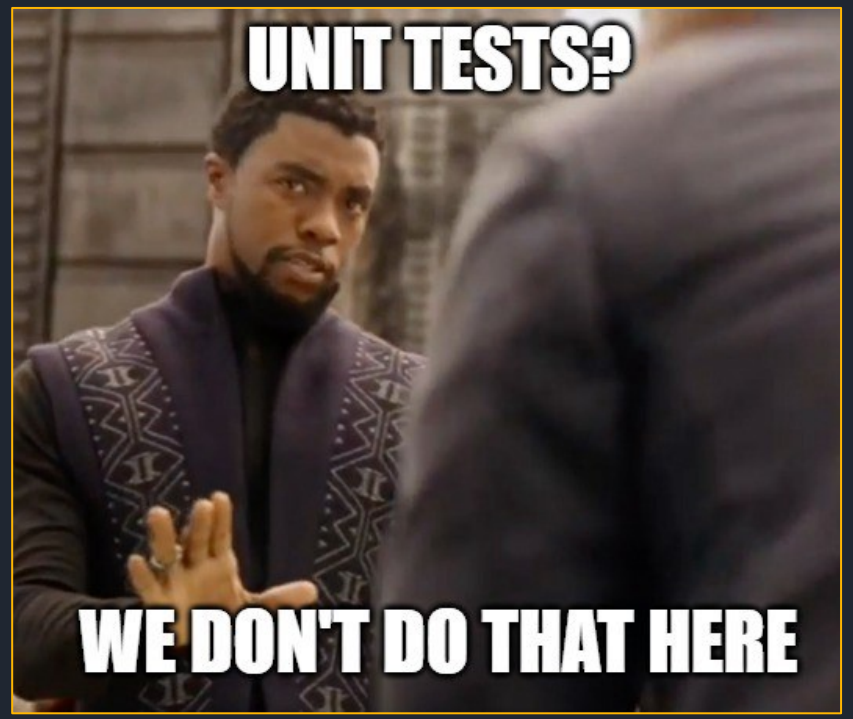
```
1  <munit:test
2      name="https-test-suite-httpsBinlowTest"
3      expectedErrorType="HTTP:UNAUTHORIZED">
4    <munit:behavior>
5      <munit-tools:mock-when
6          doc:name="Mock when"
7          processor="http:request">
8        <munit-tools:then-return>
9          <munit-tools:error typeId="HTTP:UNAUTHORIZED" />
10       </munit-tools:then-return>
11     </munit-tools:mock-when>
```

Tests can be configured to expect an exception thrown.

Mock when allows to simulate error conditions in tests.

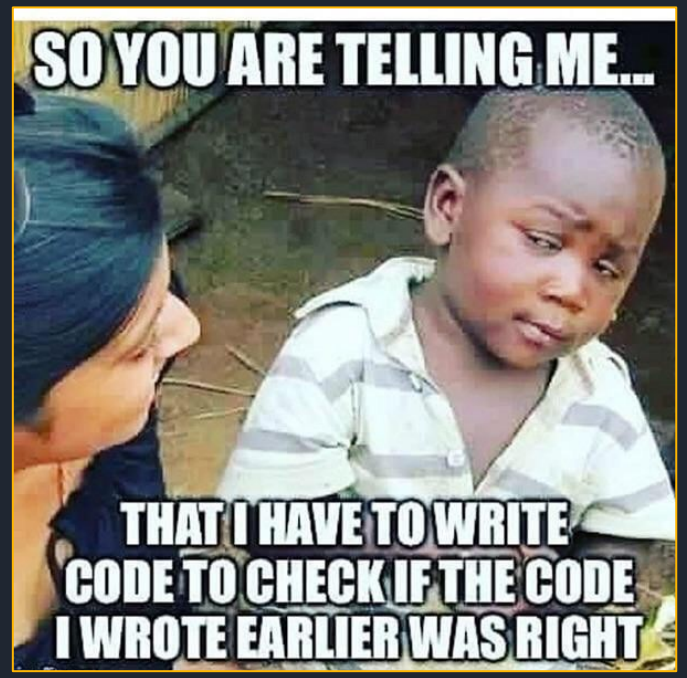
7.1 MUnit Testing

- Import new app
- Deploy & Test



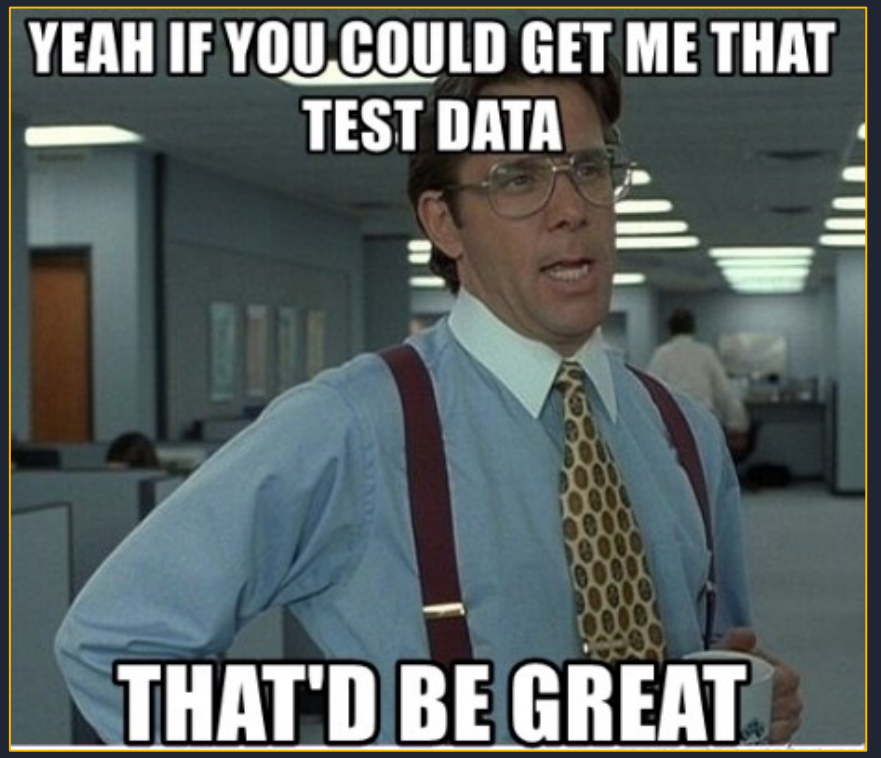
7.2 MUnit Testing

- Configure MUnit Support
- Add a sample test
- Run & Test



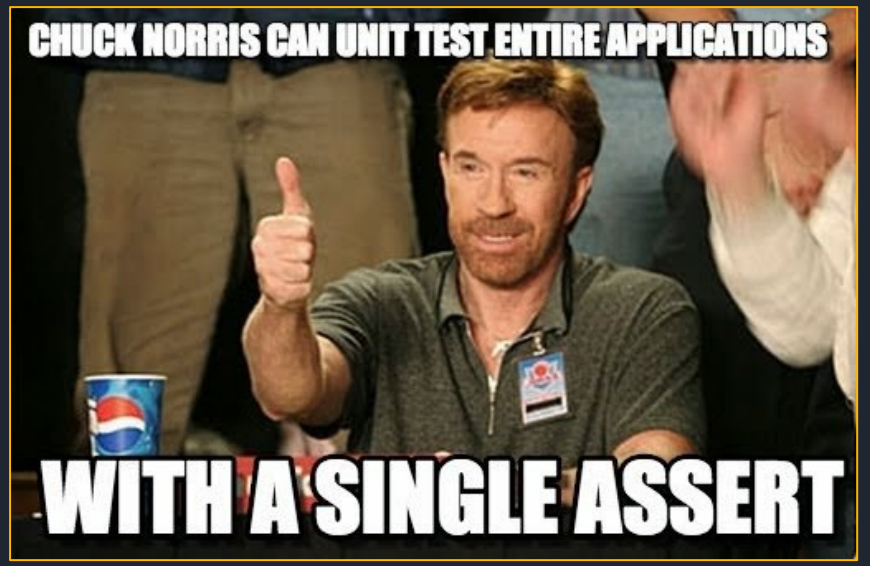
7.3 MUnit Testing

- Set Event Data
  - Set Input Payload
  - Set Query Parameters
- Run & Test



7.4 MUnit Testing

- Add Assertions
  - Check Payload is not null
  - Check Payload name field
- Run & Test

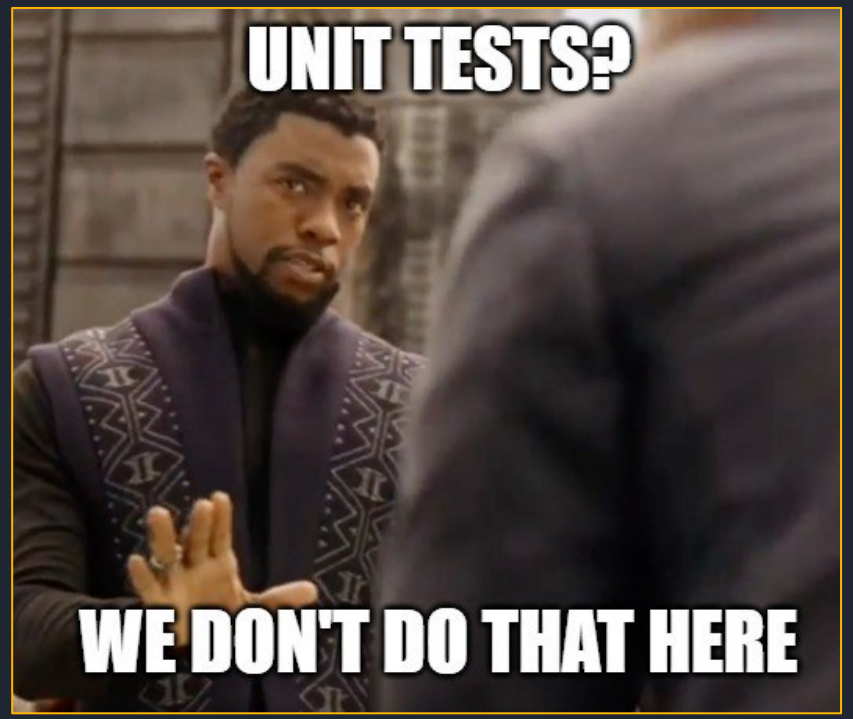


7.5 MUnit Testing

- Mock Dependencies
  - Mock payloads
  - Mock all external services
- Add a new Payload Assert
- Run & Test

7.6 MUnit Testing

- Verify
  - All external services are called once
- Run & Test



7.7 MUnit Testing

- Spy
  - Assert the payload
    - before invoking external service
    - after invoking external service
- Run & Test

7.8 MUnit Testing

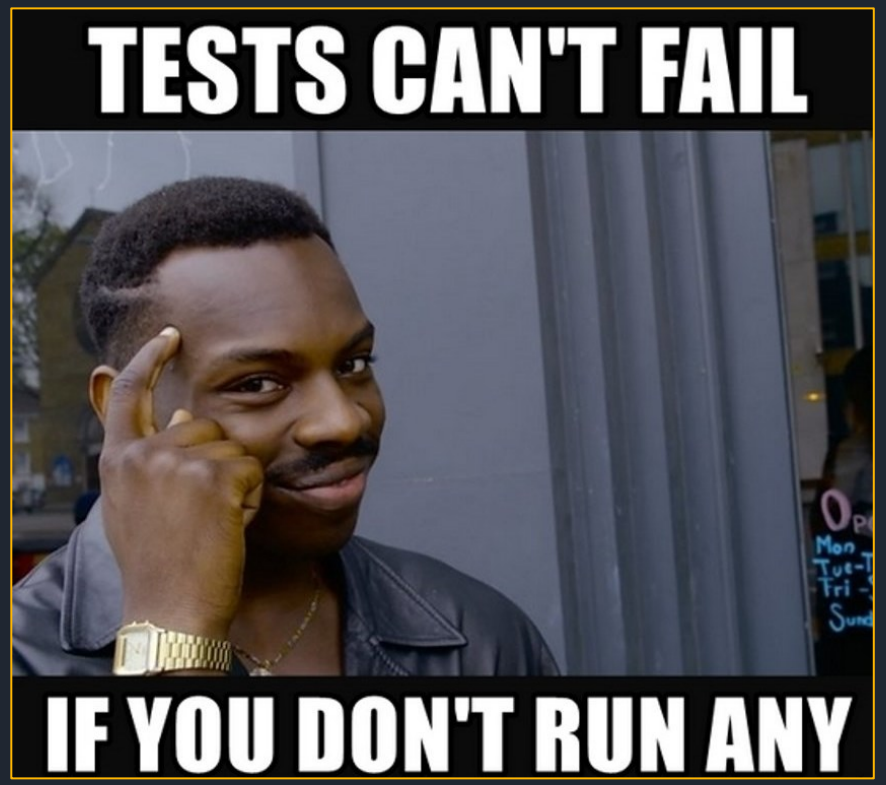
- Refactoring Munit Tests and Data

7.9 MUnit Testing

- Tests Flow with custom error handler
- Mock Errors

MUnit Testing

- Maven MUnit Testing
- Maven Commands
- Various Configurations
- Coverage reports



MUnit using Maven *(this config doesn't apply for Studio)*

mvn clean test	mvn clean package -DskipTests	mvn clean test -Dmunit.tags=<munit-tag>

mvn clean test -Dmunit.test=<regex-test-suite>

mvn clean test -Dmunit.test=<regex-test-suite>#<regex-test-name>

runtimeProduct	MULE or MULE_EE
	if app has enterprise connectors/config we need to use MULE_EE for MUnit testing *(example – Mule Secure Properties Config)*
requiredApplicationCoverage	The overall coverage of all your application
requiredResourceCoverage	The coverage level of each individual Mule configuration file
requiredFlowCoverage	The coverage of event processors in each flow
failBuild	if **false**, and the coverage levels are not reached,

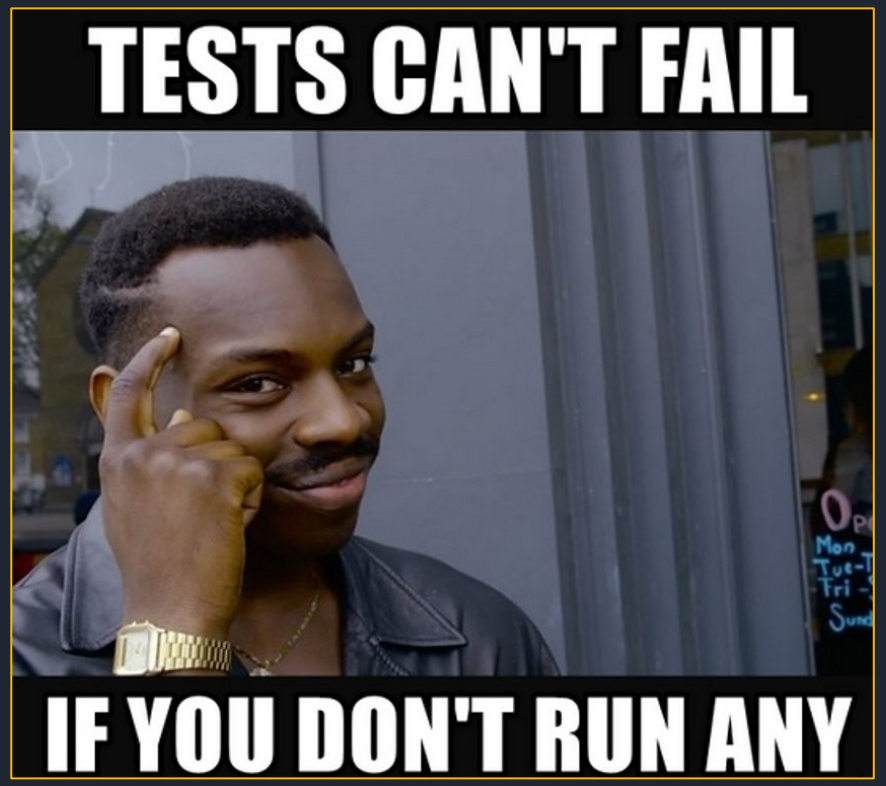
```
MUnit Coverage Summary                                                    .
* Resources: 3 - Flows: 4 - Processors: 5
---------------------- WARNING ----------------------------------
* Flow: file4.xml -> file4Flow2 coverage is below defined limit. Required: 50.0% - Current: 30.00%
```

Ignoring a flow or a file	Does not count as coverage data.
	Does not cause a build to fail if the flow is not tested or if the flow does not reach coverage metrics

```xml
<plugin>
  <groupId>com.mulesoft.munit.tools</groupId>
  <artifactId>munit-maven-plugin</artifactId>
  <version>${munit.version}</version>
  <executions>
    <execution>
      <id>test</id>
      <phase>test</phase>
      <goals>
        <goal>test</goal>
        <goal>coverage-report</goal>
      </goals>
    </execution>
  </executions>
  <configuration>
    <runtimeVersion>4.2.2</runtimeVersion>
    <runtimeProduct>MULE_EE</runtimeProduct>
    <environmentVariables>
      <env>exampleValue</env>
    </environmentVariables>
    <coverage>
      <runCoverage>true</runCoverage>
      <failBuild>true</failBuild>
      <requiredApplicationCoverage>75</requiredApplicationCoverage>
      <requiredResourceCoverage>50</requiredResourceCoverage>
      <requiredFlowCoverage>50</requiredFlowCoverage>
      <formats>
        <format>console</format>
        <format>html</format>
        <format>json</format>
        <format>sonar</format>
      </formats>
      <ignoreFlows>
        <ignoreFlow>flow-1</ignoreFlow>
      </ignoreFlows>
      <ignoreFiles>
        <ignoreFile>mule-config-1.xml</ignoreFile>
      </ignoreFiles>
    </coverage>
  </configuration>
</plugin>
```

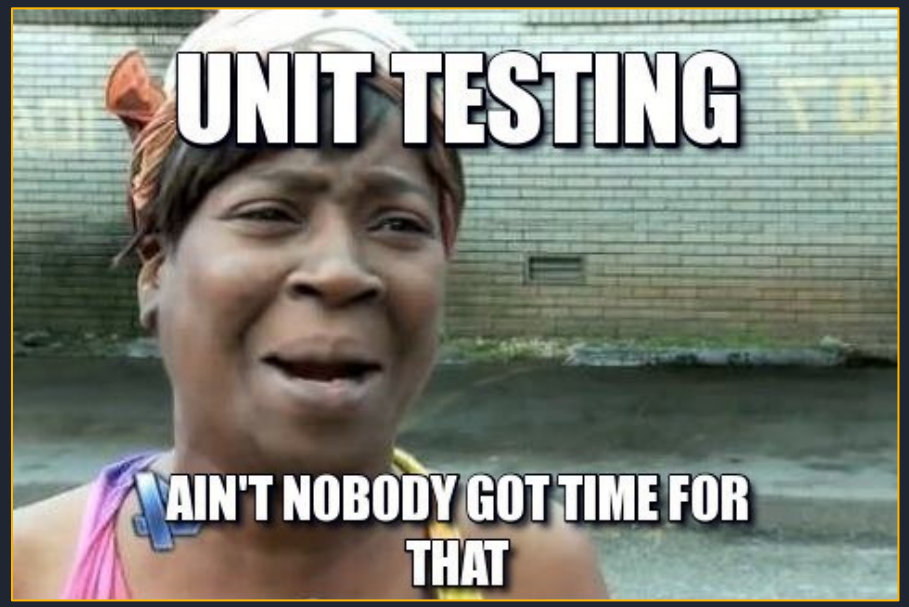
7.10 MUnit Testing

- Maven MUnit Tests
  - Failure

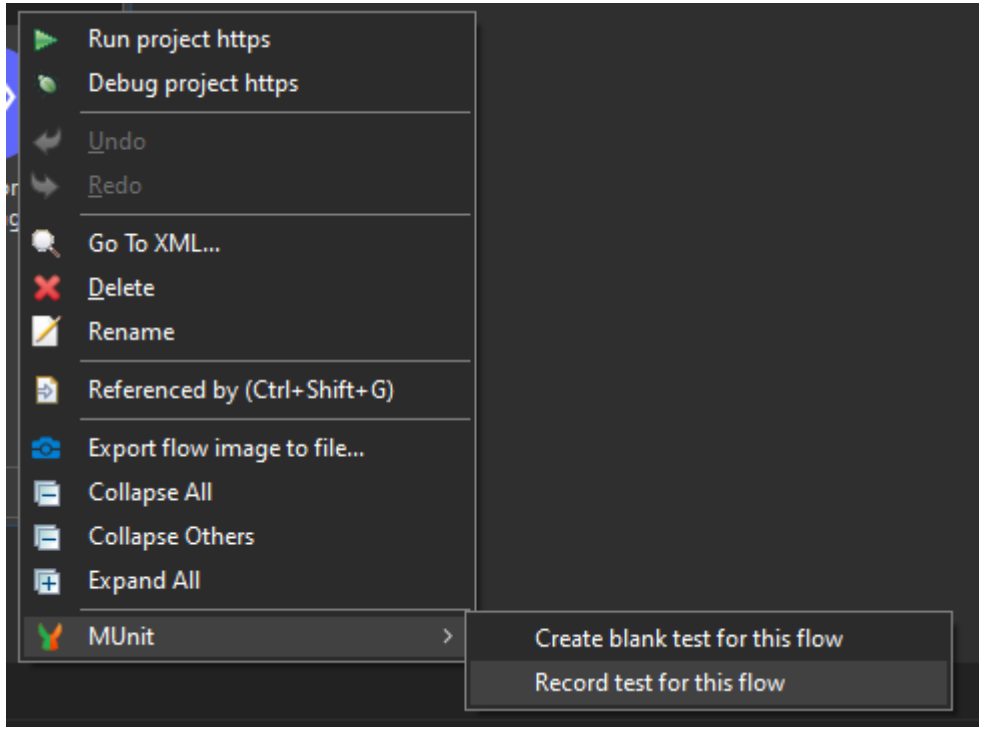


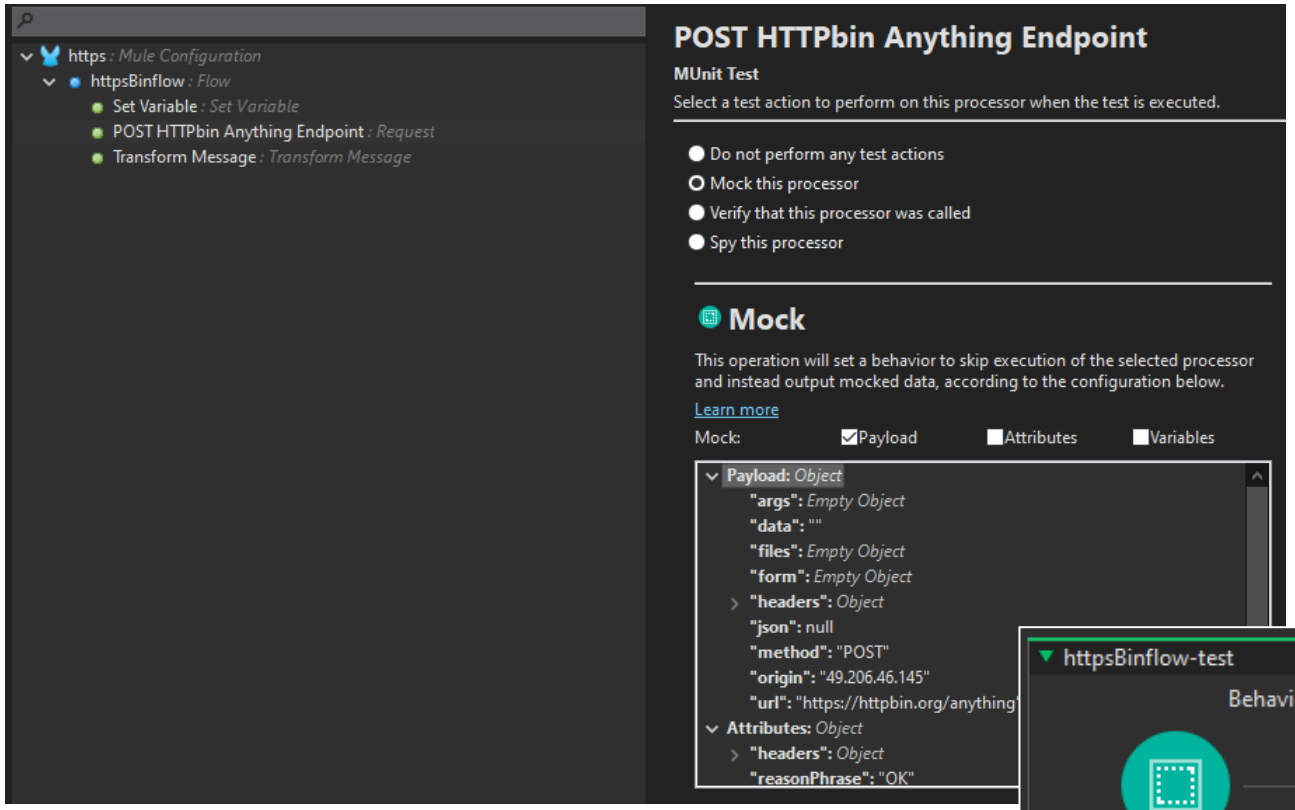
MUnit Testing

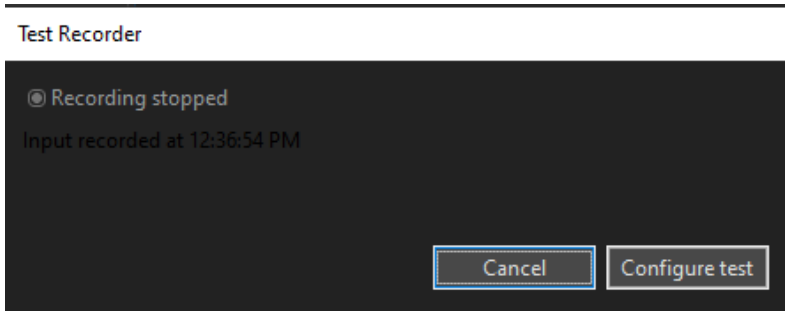
- Automated MUnit Recorder
- Configuration

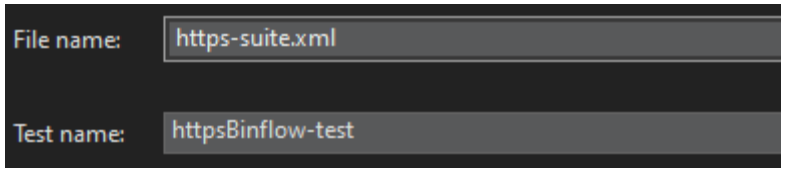


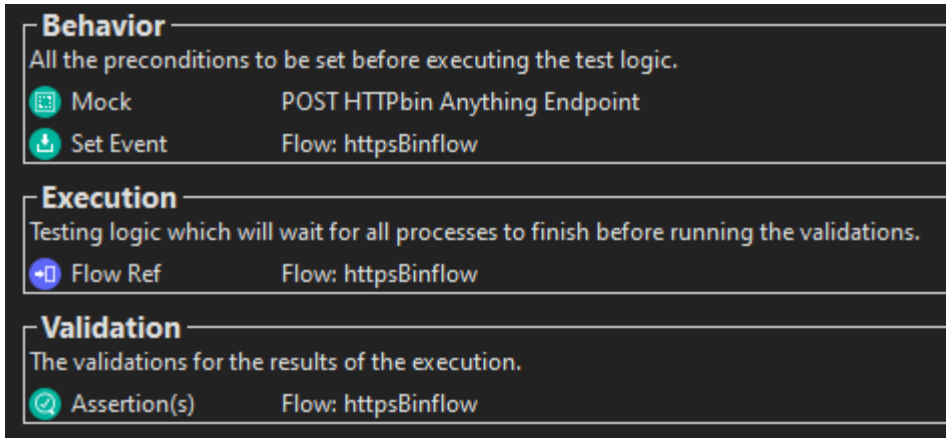
MUnit Automated Test Creation using MUnit Recorder

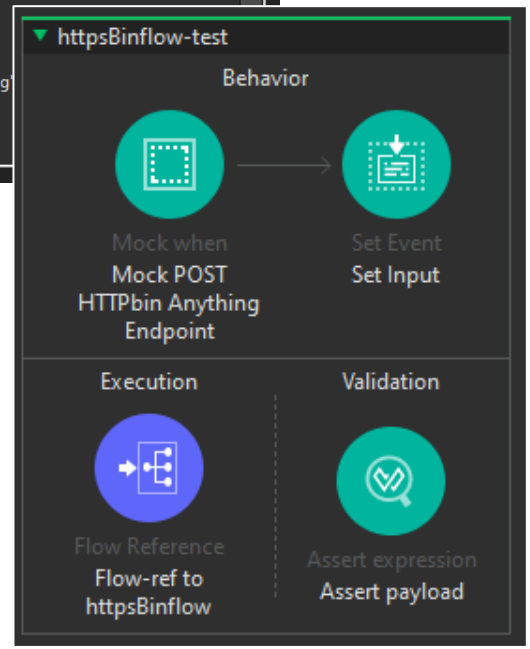






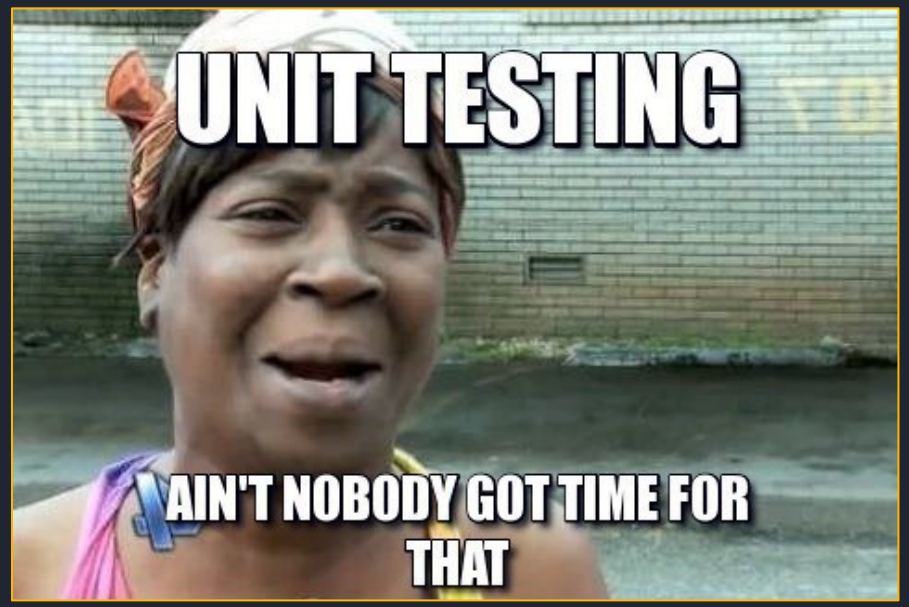






7.11 MUnit Testing

- Record MUnit Tests
- Check out the configurations
- Run & Test



Section 7 – Completed

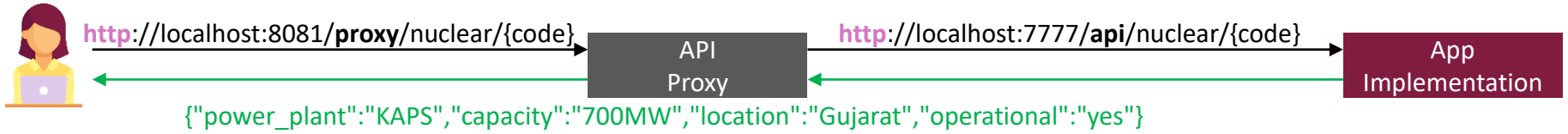
#7	MUnit Testing

- Configure MUnit Test Suite
- Set event data
- Mock Dependencies
- Assert Responses
- Verify Calls
- Spy processors
- Refactor MUnit Code
- Test Flows with custom error handlers
- Automate MUnit using MUnit Recorder

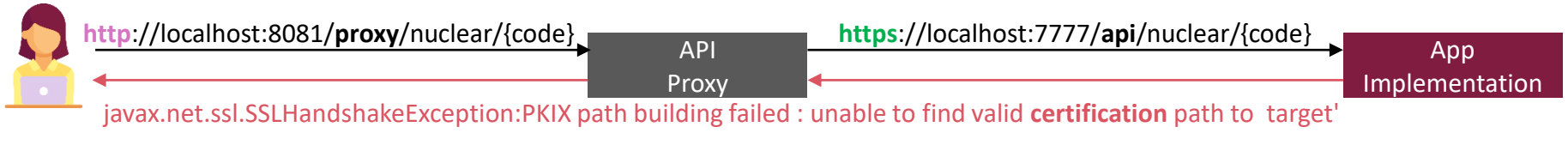
HTTPS API Proxies

- API Proxies
  - TLS
    - Inbound Traffic
    - Outbound Traffic
- Secret Groups
  - TLS Context
  - KeyStore
  - TrustStore

Build HTTPS API Proxies

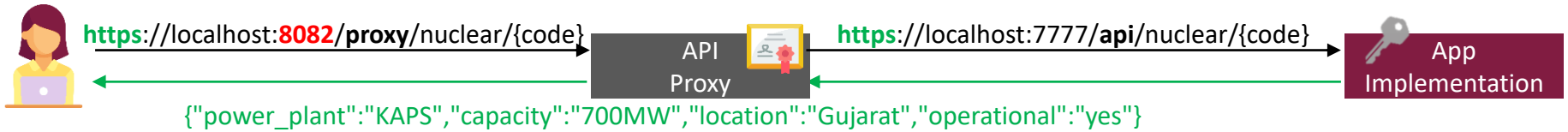



```
keytool -genkey  -alias app \
-keysize 4096  -keyalg RSA  \
-keystore app-keystore.jks
```

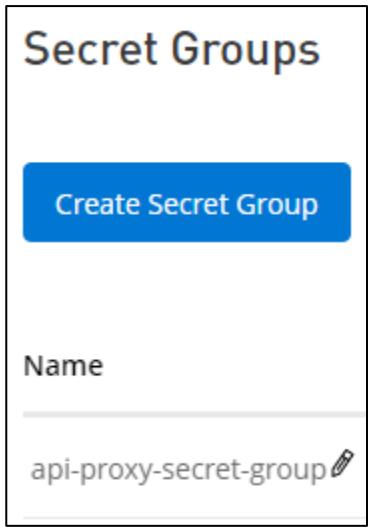


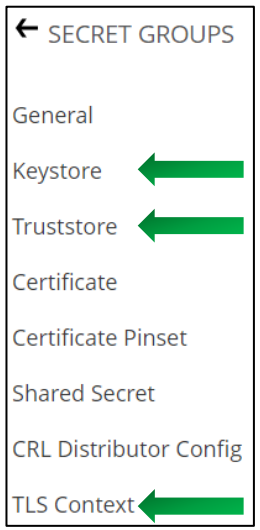

```
keytool -export -alias app \
-keystore app-keystore.jks  \
-file app_public.crt
```

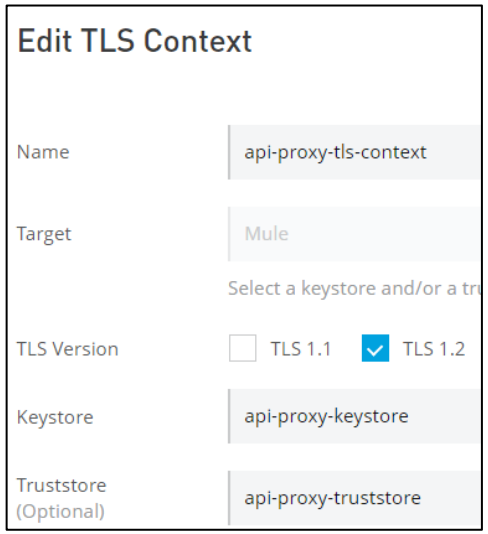

```
keytool -import -alias proxy \
-keystore proxy-truststore.jks \
-file app_public.crt
```

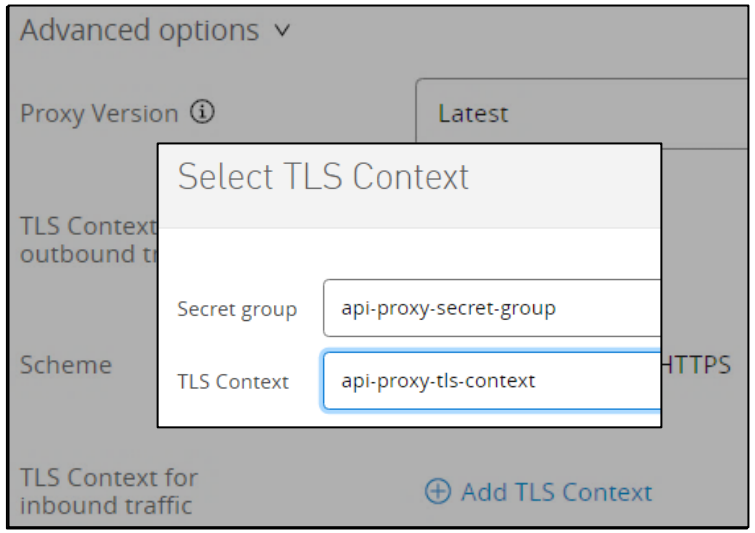


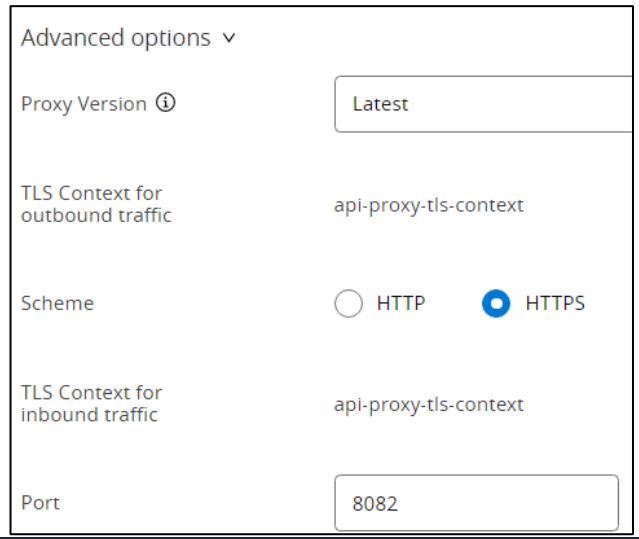

```
keytool -genkey  -alias proxy \
-keysize 4096  -keyalg RSA  \
-keystore proxy-keystore.jks
```











A-1 HTTPS API Proxies

- API Proxies
  - TLS
    - Inbound Traffic
    - Outbound Traffic
- Secret Groups
  - TLS Context
  - KeyStore
  - TrustStore