

MULESOFT CERTIFIED DEVELOPER LEVEL 2 PREPARATION COURSE



Hello and welcome to the Mulesoft certified developer level 2 preparation Course.
In This course we will go through all the concepts with hands-on exercises

whoami

MCD2

- Siddharth Barahalikar
- Experience in Development and Management of APIs through Apigee, Mulesoft, WSO2
- Apart from APIs, I work on Openshift, Jenkins, Kubernetes, Docker, Nodejs, DevSecOps Practice
- GitHub – <https://github.com/sidd-harth>
- LinkedIn – <https://www.linkedin.com/in/barahalikar-siddharth>
- Blogs – https://www.hcltech.com/blogs/profile/siddharth-b_323667



CERTIFICATIONS

API	API MANAGEMENT	INTEGRATION	DEVOPS	CLOUD

IT & Software > Other IT & Software > DevSecOps

DevSecOps - Kubernetes DevOps & Security

Learn how to integrate & monitor security within Applications, Docker, Kubernetes using DevSecOps/SecOps

Bestseller 4.7 ★★★★☆ (152 ratings) 1,903 students

Created by Siddharth Barahalikar

barahalikar.siddharth MuleSoft Certified Developer - Level 2 (Mule 4) Preparation

My name is Siddharth and I will be your instructor for this course.

So about me?

I am a Technical Architect and specialise on API Management and DevOps Technologies. For most part of my career I worked on various API Management tools like Google Apigee, WSO2, Layer7 & MuleSoft. I also work on DevSecOps and Kubernetes.

And these are some of the badges I have acquired in the last few years and the latest one is MCD Level 2 certification.

This is my 2nd online course, my 1st course was on DevSecOps which is one of the best seller and top rated course.

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

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:

Pre-requisites

- MuleSoft Certified Developer - Level 1
- MuleSoft Certified Platform Architect - Level 1
- MuleSoft Certified Integration Architect - Level 1

Topics

- 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

barahalikar.siddharth MuleSoft Certified Developer - Level 2 (Mule 4) Preparation

This course doesn't talk about what mulesoft is or what it is used for, because if you are preparing for this Level 2 certification, you already know about it.

As we know, there are 3 major Level 1 Mule certifications, which are MCD L1, MCIA L1, MCPA L1

This course is specifically designed for MCD L2 certification which is all about working independently on production ready mule apps.

It has 5 major topics

And

These 5 topics have around 30+ subtopics

We will dive into each of these topics, by first understand the concept and then implementing it

So how do we do this?

Course Details

MCD2

Course Structure

- Theory - Slides
- Demos
 - Step-by-step instructions

Pre-requisites

- MCD (Mule 4) – Level 1
- Mule 4 Fundamentals
- Basic Maven Concepts

Walkthroughs

- Code Snippets
 - GitHub Repo
 - Download Material

Practice Test

- Questions
 - Mimic the real exam
 - Explanation & Links

API-led Architecture & Artefacts

MCD2

Super Organization

Super Hero Movies | Super Hero BioData

Super Tickets Booking Service

barahalikar.siddharth

MuleSoft Certified Developer - Level 2 (Mule 4) Preparation

barahalikar.siddharth

MuleSoft Certified Developer - Level 2 (Mule 4) Preparation

Course Structure –

The course is structured in a way that, I present the topics in a simple and visual way using slides which are followed by hands-on walkthroughs. I would recommend you to do these walkthroughs to get a deeper understanding of the topic

Pre-requisites – Talking about the pre-requisites,

It is highly recommended to complete the Mule 4 level 1 developer certification, before going through these topics.

If not,

It would be great if you have completed the Mule 4 developer fundamentals course, because we will not cover MuleSoft from scratch.

We will cover Maven but Basic Maven knowledge is always good to have.

Walkthroughs

All code snippets and walkthroughs will be provided to you via a Github repository or as a downloadable resource.

Practice Tests

At the end of the course, I had planned to include a couple of practise exam, that helps you assess course content.

For the time being this is kept on hold because MuleSoft's MCD Level2 Practice exam is excellent and that should be sufficient to crack the certification.

Course Objective

MCD2

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
barahalikar.siddharth	MuleSoft Certified Developer - Level 2 (Mule 4) Preparation

To cover all the topics, we have divided this course into 8 Sections starting from 0

Section 0

We will start by setting up the local environment by install JAVA, Maven, Anypoint Studio and deploying the required backend services to CloudhUb after importing and publishing RAML in anypoint platform

Section 1

In Section 1 we will setup the mule project following best practise, add deployment strategy, maven configuration, Modularize Mule app using parent-pom, add secure properties and configure one-way tls

Sections 2

In Section 2 we start adding integration logic, hit multiple APIs in parallel, manage non-functional requiremens, handle errors and cache the data to improve app performance

Section 3

In Section 3 we add HTTP callback and process the messages asynchronously using

VM and Anypoint MQ connectors. We explore circuit breaker, Operational Tracing, Manage correlation id and do schema validations

Section 4

In Section 4 We mention the Mule applications using health endpoints. Use Shared library and create new connectors using XML SDK

Section 5

In Section 5 We develop a custom API policy from scratch and apply it both offline and online

Section 6

In Section 6 We understand how to use TLS config for SOAP webservice and consume a service using Mutual TLS Authentication.

Section 7

In Section 7 We configure and write Munit test cases for our Mule apps by using various MUNIT Connectors.

We only see a small subset of topics in this slide. We will be covering a lot more topics in detail within each section.

Setting up Expectations

MCD2

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

barahalikar.siddharth

MuleSoft Certified Developer - Level 2 (Mule 4) Preparation

Before we start, I would like to setup some expectations right at the beginning.

By default all topics are covered with handson excercises, with few exceptions.

Anypoint MQ is not available in Anpoint Platfrom trail accounts. I have used Anypoint MQ in this course but I cannot share it's details and so you will not be able to do Anypointmq HANDSON excercises. We will provide slides, videos, code snippets

and it is the same for Functional Monitoring. So we will be using an external API Monitoring solution to understand the concepts, which is more or less the same.

We do not have and will not provide Mule Enterprise Maven Repo credentials. Munit maven execution depends on this and hence we cover the concept through slides and do Munit testing using Anypoint Studio

All other hands-on walkthroughs can be done without Enterprise credentials.

If you have access to any of these resources via your partner portal or some other means, you could use them for handson purpose. Please do not use your customer/production account.

MULESOFT CERTIFIED DEVELOPER LEVEL 2 PREPARATION COURSE



linkedin.com/barahalikar-siddharth



mcd-level2.slack.com

Do check out my LinkedIn profile and slack workspace for new updates, offers and for any other queries.

Feel free to look at the course description and I look forward to see you inside

~~THANK YOU~~

Section 0

MCD2

#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

barahalikar.siddharth

MuleSoft Certified Developer - Level 2 (Mule 4) Preparation

In Section 0, we will install the software
Understand the course usecase and the architecture
And setup the backend services by

And all these startup files will be provided thorough a git repository

0.1 Software Installation

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



MCD2

In this session we install the softwares

barahimkarandam

Setup Local Environment

MCD2

OpenJDK8

maven
3.6.x or later



7.11.1 or later

Mule 4
Standalone



SoapUI

The SoapUI logo icon is a green stylized 'S' inside a circle.

barahalikar.siddharth

MuleSoft Certified Developer - Level 2 (Mule 4) Preparation

For doing the handson walkthroughs, these softwares needs to be installed and configued on your machines.

Make sure you dowanlod the correct version.

These installtions are straight forward, so I would leave them to you. In the attached material, I will provde the links and docs for your reference.

Usecase Architecture

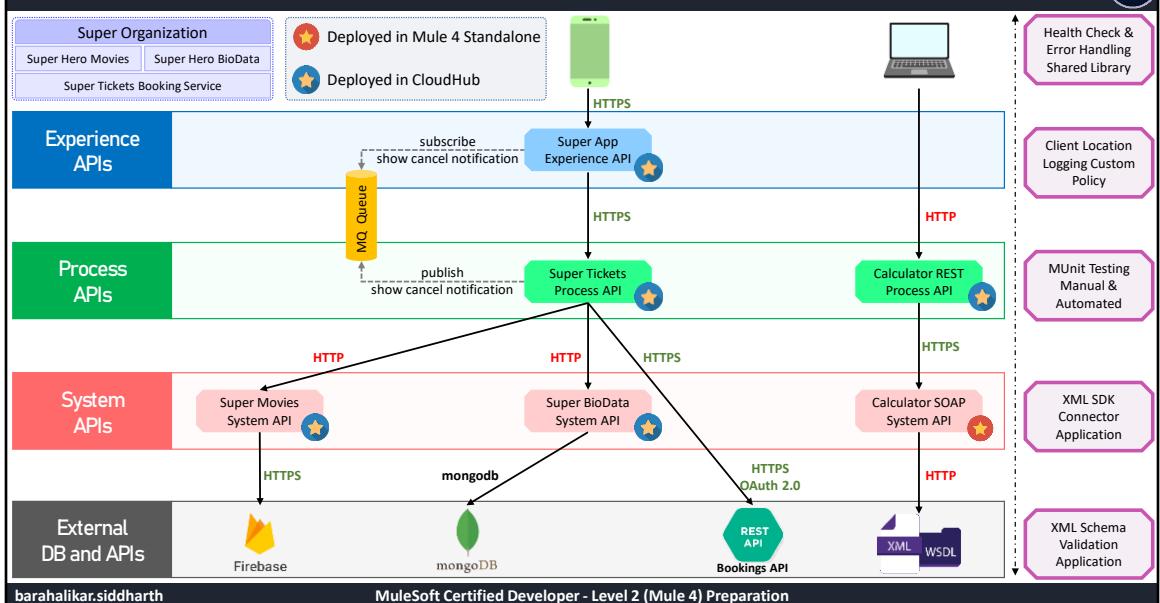
- API-led Architecture
- Other Artefacts used in course



barahatkaranthash

In thi ssession, we will have a look at the course usecase followed by it's architecture. We will also see what other artefacts will be developed as par tof this course

API-led Architecture & Artefacts



Lets assume,

We have a organization named Super Organization, which holds the data for Super hero movies and super hero biodata.

Whenever a new movies releases, they also provide a Ticket Booking Service to the end user via a user application.

The users can book tickets, check the booking status and show cancellation status on this applicaton.

The data is spread across 3 external services – Firebase, MongoDB and a REST API
We develop 2 SAPI (Super Movies and Super BioData) to unlock the data from these core systems

We then develop a Super Tickets PAPI to interact with both the SAPI and the 1 external REST API service

This PAPI is accessed through a EAPI, which also subscribes to the Anypoint MQ Queue to get show cancel notifications published by the PAPI

For mutual authentican topic,

We also have an external WSDL service which is unlocked by SAPI. This SAPI is in turn consumed consumed by a PAAPi using mutual authentication. this use case

showcases Mutual authentication between two Mule Application

For deployments,

The SOAP SAPI is deployed to a Mule 4 standalone server

All others arteafcts are deployd to Clouduhub

For other topics,

We also work on various other application, librareies and connectors in this course

And for the final note – this is a very simple straightboard usecaces which covers all the topics of this course,, please do not try to find loophole in it

0.2 Anypoint Platform

- Create a new Account
 - Assign Environments



MCD2

In this session we install the softwares

barahamercanthan

0.3 API Specification

- Import RAML
- Publish RAML
- Explore Exchange



barahamkaranthurth

MCD2

0.4 Anypoint Platform

- API Manager
 - Create API Instance
 - Add Automated Policy



MCD2

barahamkaranthan

In this session we install the softwares

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



MCD2

In this session we install the softwares

0.6 Access Applications

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



MCD2

In this session we install the softwares

barahimkarandam

Section 1

MCD2

#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

barahalikar.siddharth

MuleSoft Certified Developer - Level 2 (Mule 4) Preparation

Welcome to Section 1, in this section we will start by adding an Open API Specification to Design center,

Configure API instance for it in API manager and Import it to Studio

Within Studio, we will add best coding conventions and setup the project structure. We will Learn about Cryptogrpahy and how to use it for secureing communicatons/data in transit.

For static data we will use Secure and Hidden properties to encrpt and mask them

We will also understand Cloudhub architecture and setup a Maven based deployment strategy.

We will also modularize mule apps using parent-pom and bill of materials concepts. We will end this section by visualizing the API interactions within Anypoint Visualizer

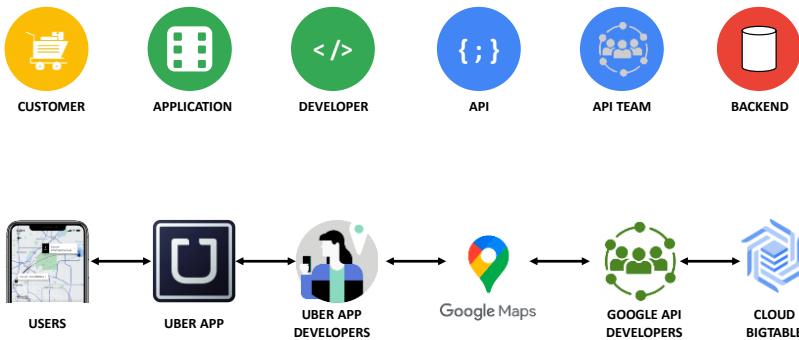
API Lifecycle

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



barahamercanthanh

API



An API is a set of rules which defines how 2 applications interact with each other.

In simple terms,

An API is a contract between 2 applications

Generally speaking we have an API/Integration team working on various backends to develop an API

These APIs are then used by APPLICATION developers to create application, these could be internal or external developers.

Then the application is consumed by the user.

A real time example could be of Google Maps,

Google stores the location details like latitude longitude details within a cloud bigtable

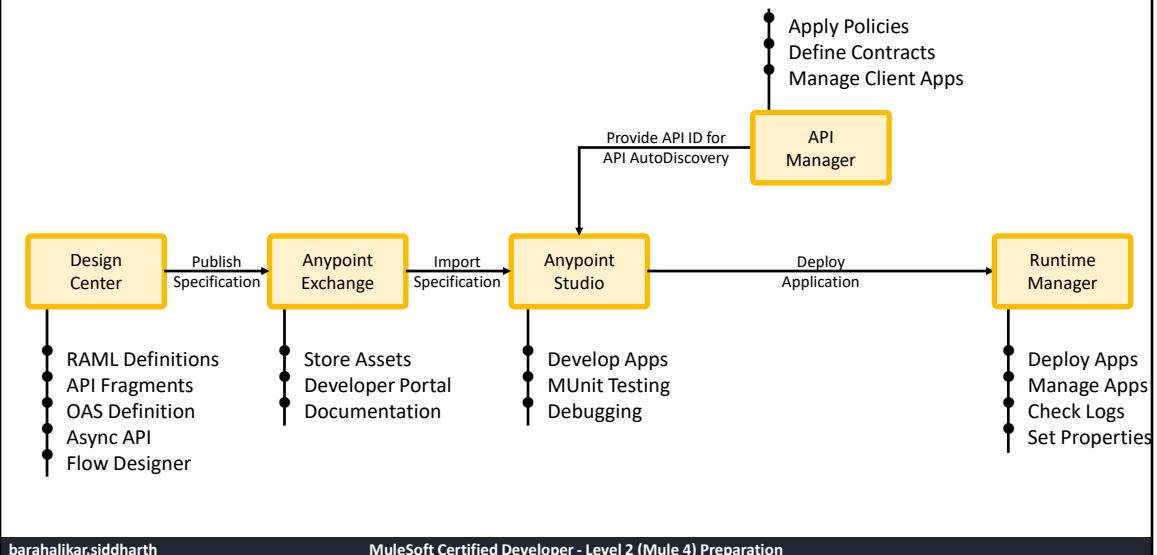
Google API developers work on this data and create Google Maps API

Now this API can be consumed by some 3rd party application developers like Uber to embed the Maps API within their application

And when the user uses the Uber app, they internally make use of the Google Maps API

API Lifecycle

MCD2



A [API lifecycle management](#) is the process of overseeing an API from its creation to retirement. This includes everything from designing, publishing, documenting, developing, securing, deploying and [monitoring APIs](#).

Design center

The first step is to design API, we make use of Design center to create APIs using RAML/OAS definitions. We Mock them and test them before publishing to Exchange

Anypoint Exchange

Is like a repository to hold all your Mule artifacts. We can add documentation to APIs, expose them to Public via a developer portal.

Studio

API Development is done using Anypoint Studio, where all the business logic like orchestration, routing, error handling is done. Once it is built it goes through Munit testing to ensure that it is functioning as expected.

Runtime manager

Then the application is deployed to Clouduhub and managed via a Runtime Manager to check logs and manage app properties.

API Manager

to ensure your application is following best practices in security we apply policies to them at runtime. And Monito all traffic via an [API manager](#)

API Specification

MCD2

RAML



```
super-movies-sapi/master

1 #%RAML 1.0
2 title: Super Heros Movies Data SAPI
3 description: Super-Movies-SAPI allows you to fetch the basic movies
4 heros.
5 version: 1.0
6 mediaType: application/json
7 types:
8   movies-data: !include movies.datatype.raml
9
10 securitySchemes:
11   basic-auth: !include exchange_modules/org.mule.examples/basic-auth-
12     basic-auth-securityscheme.raml
13 securedBy: basic-auth
14
15 /movies/{id}:
16   uriParameters:
17     id:
18       required: true
19       example:
20         value: MARVEL0001
21     get:
22       description: This resources fetches bio data based on the unique
23       Parameter
24       responses:
25         200:
26           description: If **id** exists, returns the bio data
```

```
Super Ticket PAPI/master

1 swagger: '2.0'
2 info:
3   title: Super Ticket PAPI
4   description: Super Ticket PAPI allows you to check the movie ticket
5   version: v1
6   host: super-ticket-papi.us-e2.cloudhub.io
7   basePath: /api/v1
8   schemes:
9     - https
10  paths:
11    /tickets/{BID}:
12      x-amf-description: Checks the passenger in with given number of ba
13      get:
14        parameters:
15          - name: BID
16            description: Unique Booking ID
17            required: true
18            in: path
19            type: string
20          - name: CID
21            description: Unique Character ID
22            required: false
23            in: query
24            type: string
25        responses:
26          '200':
27            description: If the BID is correct, returns ticket booking d
Super Movies SAPI for the character data will be returned.
```

barahalikar.siddharth

MuleSoft Certified Developer - Level 2 (Mule 4) Preparation

API specifications can be defined in multiple ways, one of them is RAML

RESTful API Modeling Language (RAML)

- o Based on YAML
- o Human readable format
- o Developer friendly
- o you can Reuse code using Fragments

Mulesoft also supports OAS,

- o Formerly called Swagger (OAS 2.0)
- o Open standard to define RESTful API interfaces in YAML or JSON format
- o it has a Wider adoption rate compared to RAML, but
- o Files tend to get bigger because of Less code sharing options

1.1 OAS Definition

- Import OAS to Design Center
- Publish to Exchange



barahamercardorath

MCD2

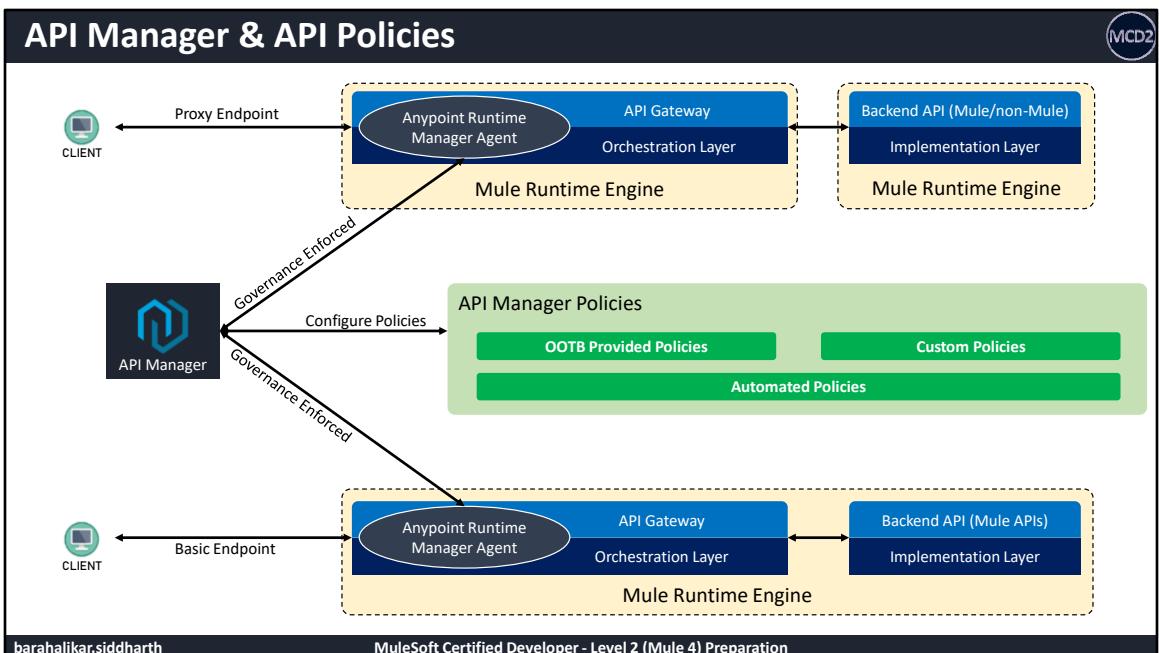
API Manager

- Proxy Endpoint
- Basic Endpoint
- API Gateway



**PUT AN API IN API MANAGER
AND ACCESS APIs USING API**

brahma&carinthia



Anypoint API Manager enables you to manage, govern, and secure APIs.

Using API Manager, we can configure various OOTB and custom policies in automated or manual way. These policies are

- ○ Applicable to all types of HTTP/1.x APIs , but
 - Does not support WebSocket APIs or HTTP/2 APIs

Lets assume we have an Mule API configured with an corresponding Autodiscovery element. To enforce policies it makes use of API Gateway.

Mule Runtime includes an embedded API Gateway.

An API Gateway allows you to add a dedicated orchestration layer which sits in front of your backend APIs.

API Manager leverages the capabilities of API Gateway which enforce policies, collect analytics data, manage proxies and authentication.

But how does API Gateway communicate with API Manager?

Anypoint Runtime Manager agent is a Mule plug-in that enables communication with

Mule runtime engine.

API Gateway uses this agent to communicate with registered API instances in API Manager.

Clients access the app using the BackendAPI Cloudbus URL

Lets assume we have another API that is not based on Mule Runtime, or an Mule app which doesn't have a autodiscovery element. To enforce policies on it we can use an API proxy which is a preconfigured Mule application deployed separately. This application is nothing but the API gateway.

Client access the app using the proxy url

1.2 API Manager

- Create API Instance
- Add Logging Policy

PUT AN API IN API MANAGER

AND ACCESS APIS USING API

barahamkarandikar

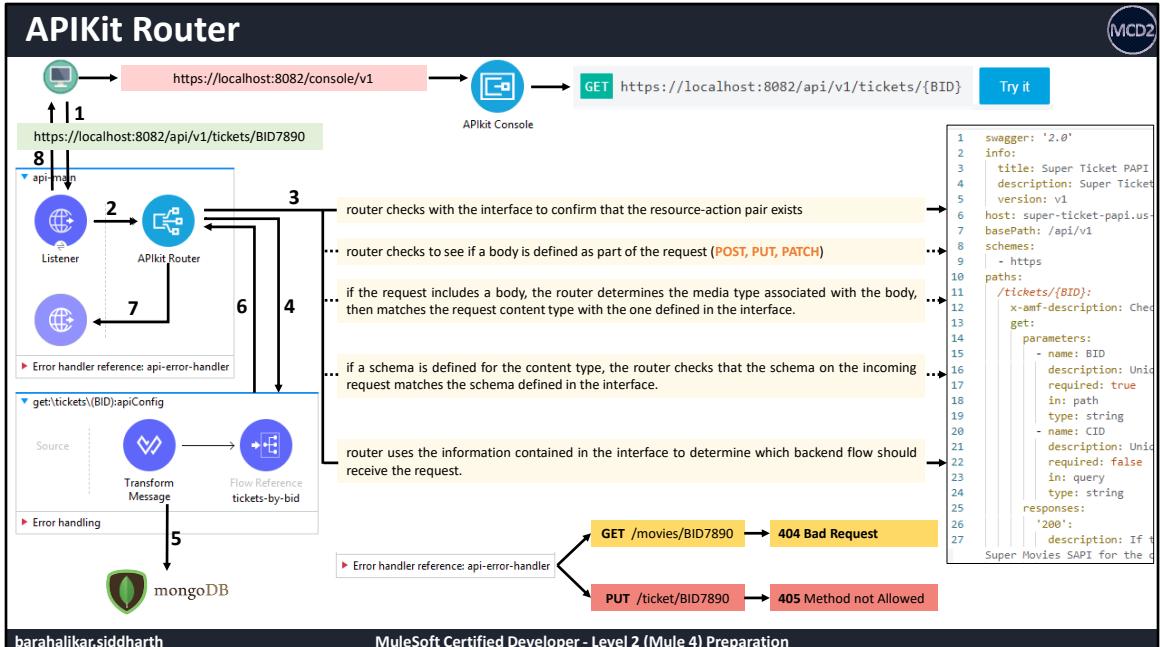
MCD2

APKit Router

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



barahmierandmih



Lets assume we have an RAML or OAS Specification published to exchange. When we import it to our Mule Application in Anypoint studio, it is going to auto generate flows based on the specification.

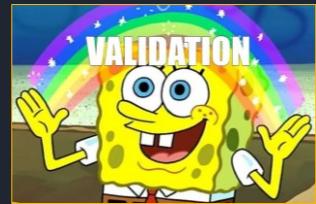
We will see how an APIkit router listen for incoming request, do validations, route to flows and returning the response

1. The end user sends an HTTP request to the API.
2. The HTTP endpoint in the main flow receives the request and passes the message to the APIkit Router.
3. The router checks with the interface to confirm that the resource-action pair exists in the interface.
 1. If it is a GET method, the router uses the information contained in the interface to determine which backend flow should receive the request.
 2. If it is a POST/PUT/PATCH method The router checks if a body is defined as part of the request.
 3. If the request includes a body, the router determines the media type of the body, then matches the request content type with the interface.

4. If a schema is defined for the content type, the router checks that the schema on the incoming request matches the schema defined in the interface. If the schema is not valid, the application rejects the request.
5. Once the flow is determined
 1. The router sends the request to flow.
 2. Flow processes the request, accessing a backend services as required.
 3. Flow returns a response to the router.
 4. The router pushes the response to the HTTP endpoint.
 5. The HTTP endpoint sends the response to the end user.
6. It also has a console endpoint, which can be used to simulate API calls by submitting requests through the Web user interface.
7. Generic error handlers are also generated to handle invalid requests

1.3 APIKit Router

- Import API from Exchange to Studio
- Deploy the App

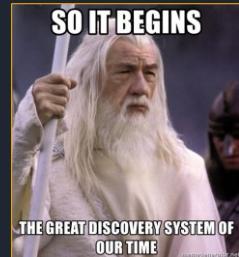


barahmarandanth

(MCD2)

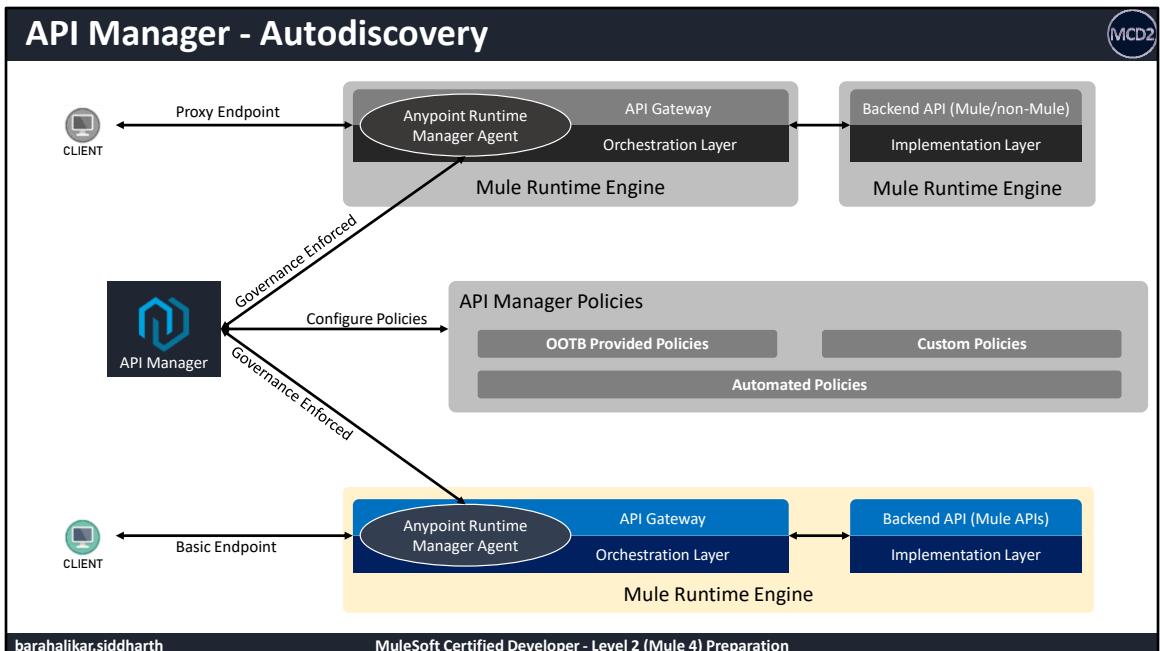
API Autodiscovery

- What is AutoDiscovery
- Steps to achieve it



barahillercardinanth

In an earlier session we talked about API Manager and API Policies. In this video we will talk about Autodiscovery



API Autodiscovery is a mechanism that manages an API from API Manager by pairing the deployed application to the API.

Configuring autodiscovery allows a deployed Mule application to connect with API Manager to download, manage policies and to generate analytics data.

We need to have an unique API Instance ID to map an API with Autodiscovery.

Using API Autodiscovery option, we do not need an extra API Proxy deployed to enforce policies. Mule applications act as their own API proxy.

A couple Point to be noted -

Multiple Autodiscovery elements pointing to an same API instance ID are not supported.

For each Mule runtime version for an API instance, we can have only one Autodiscovery element pointing to it.

API Autodiscovery Steps

MCD2

Get API Instance ID

Mule runtime version: 4.4.0-2021122

API Instance ID: 17634014

Label: super-ticket-papi-prod

Get ENV's Client ID & Secret

Environment name: prod

Environment ID: f853a096-9f5

Client credentials

Client ID: f72ff9342403

Client secret:

Add Env Credentials in Studio

Environment Credentials

Client Id: f72ff93424

Client Secret:

Validate Environment Credentials: Validate ✓

Organization Name: -

Run local or Deploy CloudHub

- ▶ Run project super-tickets-papi
- ▶ Debug project super-tickets-papi
- undo

Download RAML from Design Center

Publish to Exchange

Deploy to CloudHub

Configure Credentials

Add AAD Config in Studio

API Autodiscovery

Service auto-discovery configuration information.

Auto-discovery configuration	Notes	Help
Auto-discovery settings		
API Id: 17634014		
Flow Name: api-main		
<input checked="" type="checkbox"/> Ignore base path on resource level policies		

Check API Status

Super Ticket PAPI v1

API Status: Active

Implementation URL: https://super-tickets-papi.p...

MuleSoft Certified Developer - Level 2 (Mule 4) Preparation

These are the steps to enable API AutoDiscovery

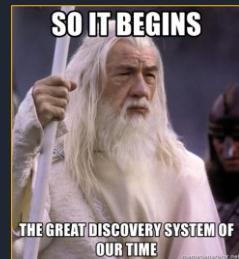
1. Get API Instance ID from API in API Manager
2. Get the environment specific client id and secret

Following the principle of least privilege, it is typically recommended to use the client ID and secret for a specific environment rather than the entire Anypoint Platform organization:

1. Configure Autodiscovery element in Studio, by adding the API ID
2. Add the Environment credentials in Studio
3. Deploy the application to Cloudhub, once deployed is successful
4. The api status on the API manager changes to Active, this indicates that the Mule App is connected the API in API Manager

1.4 API Autodiscovery

- Steps to achieve it

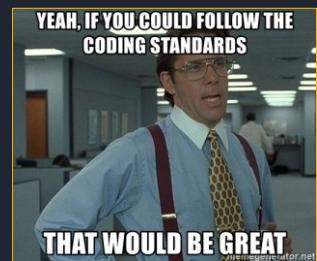


barahimercanthan

MCD2

Coding Conventions

- Naming Conventions
- Project Structure
- Anypoint Studio Settings



Coding conventions are a set of guidelines for a specific [programming language](#) that recommend [programming style](#), practices, and methods for each aspect of a program written in that language.

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	apiHttpListenerConfig	oauthTokenObjectStore	anypointMQConfig
Flow SubFlow Names	kebab-case	tickets-by-bid	compensate-movies-service	get-biodata-by-cid-service
Flow Reference Name	Name should always be equal to referred/calling flow name			

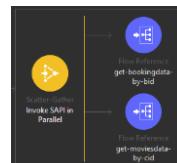
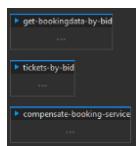
Asset name	Super Ticket PAPI
Asset ID	
	super-ticket-papi

```
    <vin:config
      name="VmConfig"
      doc:name="Vm Config"
      doc:id="beb8de80-0082-45c4-87af-bf2864ec8302">
        <vm:queues>
          <vm:queue
            queueName="show-cancelation-notification-q"
            queueType="PERSISTENT"
            maxOutstandingMessages="100" />
        </vm:queues>
    </vin:config>
```

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

super-tickets-papi < pom.xml
 src/main/mule (Flows)
 apixml
 error.xml
 global.xml
 health.xml
 main.xml

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



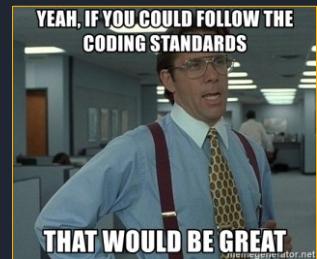
barahalikar.siddharth

MuleSoft Certified Developer - Level 2 (Mule 4) Preparation

We will be looking at naming convention of various resources,

1.5 Coding Conventions

- Naming Conventions
- Project Structure
- Anypoint Studio Settings



barahamercanthan

MCD2

Secure Communication

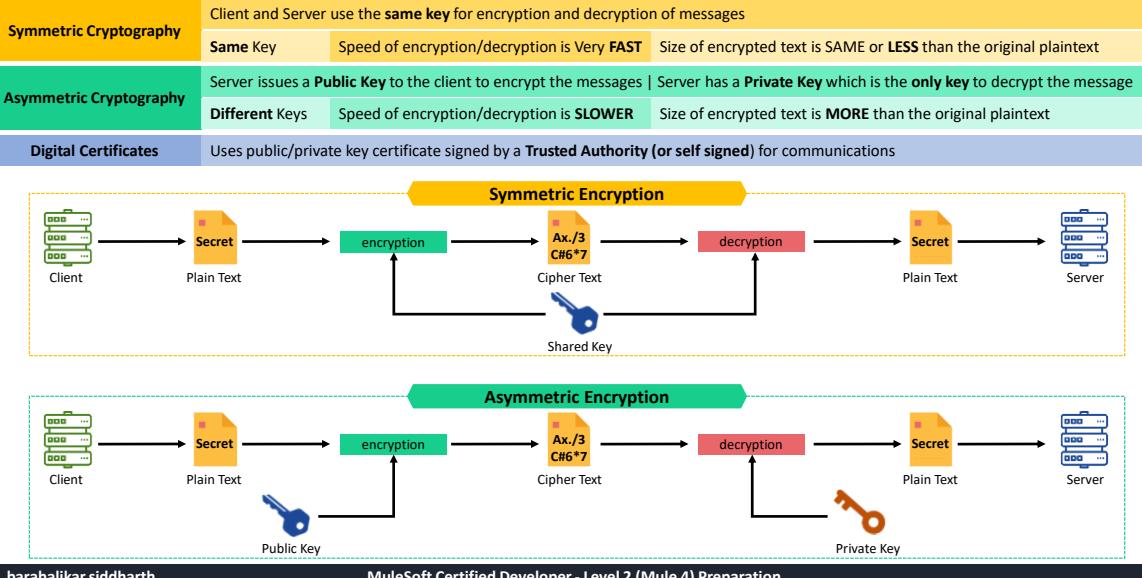
- Using Cryptography
 - Symmetric Encryption
 - Asymmetric Encryption



barahmierandforth

Secure Communication – Using Cryptography

MCD2



barahalikar.siddharth

MuleSoft Certified Developer - Level 2 (Mule 4) Preparation

Secure communication is the safe way of sharing information that cannot be intercepted by a third party users.

It is supported using Cryptography like Symmetric, Asymmetric cryptography and digital certificates

- In **Symmetric cryptography**
 - the client and server share the same key to encrypt and decrypt data
 - The key lengths varying between 128 and 256 bits and so the
 - Speed of encryption/decryption is very fast
 - Size of the text is same or less than the original plain text
- In **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
 - *So 2 different keys are used for encryption and decryption*
 - key length between **1,024 and 4,096 bits** because of which the
 - Speed of encryption/decryption is slower
 - Size of the text is **MORE** than the original plain text
 - Talking about
- **Digital Certificate – It uses** public/private key certificate signed by a **Trusted**

Authority (or self signed) for communications. We will see more about this in the next slide

- Lets have a quick look at the **Symmetric** Encryption
 - We have a client and a server
 - The client encrypts a plain text using a shared key which results in a cipher text
 - Ciphertext is **encrypted text transformed from plaintext using an encryption algorithm**
 - **The server** decryptes the cipher text using the same shared key to get the plain text
 - *One disadvantage here is, At some point the client and the server have to exchange this SHARED key for encryotion/decryption. If it get intercepted by a third party users. Then they will be able to access everything*
 - We will see how this can be overcomed in the next slide
- Lets have a quick look at the **Asymmetric** Encryption
 - The flow remains the same
 - Here the client uses a public key to encryo the plain text
 - And Server uses a Private key to decrypt the cipher text
 - I hope you understood the difference between symmetric and asymmetric encryption, in the next session we will look at di

Secure Communication

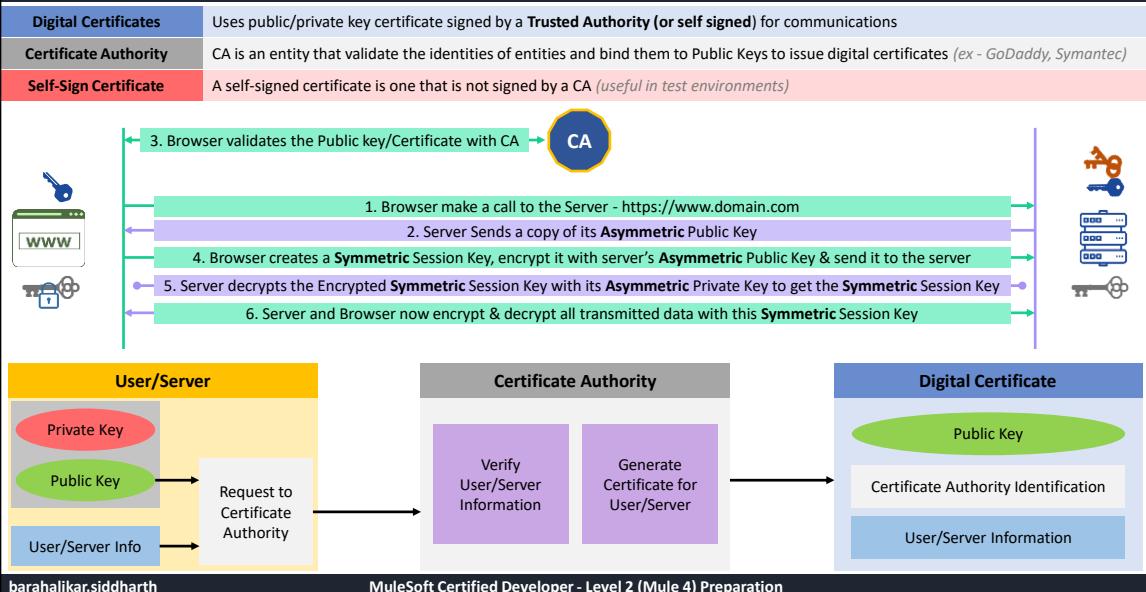
- Digital Certificates
 - Certification Authority
 - Self-signed Certificate
 - Process Steps



barahmierandmehr

Digital Certificates

MCD2



Digital certificates are electronic credentials used to assert the online identities of individuals, computers, or applications.

Similar to ID cards like passport, driving license, these certificates contain public keys & identity of the owner

Digital certificates Uses public/private key certificate signed by a **Trusted Authority (or self signed)** for communications

Certificate Authority is a company that validate the identities of entities and bind them to Public Keys to issue digital certificates (ex - GoDaddy, Symantec)

If you are testing your application, we can use self-signed certificates, A self-signed certificate is one that is not signed by a CA

What is the process of getting a Digital Certificate?

The user generates a Key pair (i.e public & Private key)

The user sends a CSR, certificate signing request to CA by passing the public key and user/server information

*The certificate authority verifies the User/Serve info, by contacting/checking them
If everything is correct, CA generates a Certificate for the Server*

*This digital Certificate contains the Server Public key, Certificate Authority
Identification Information and the Server Information*

*Once the server has the keys, how do they establish a Secure Connection with the
client?*

*Please note that at the end of the process, both client and server are using a
SYMMETRIC KEY to encrypt/decrypt the data. But to exchange this SYMMETRIC KEY
securely we have use ASYMMETRIC Keys*

I hope this make sense and know you are an expert in SSL. Thank you

SSL/TLS Options

- One-way TLS
- Two-way TLS



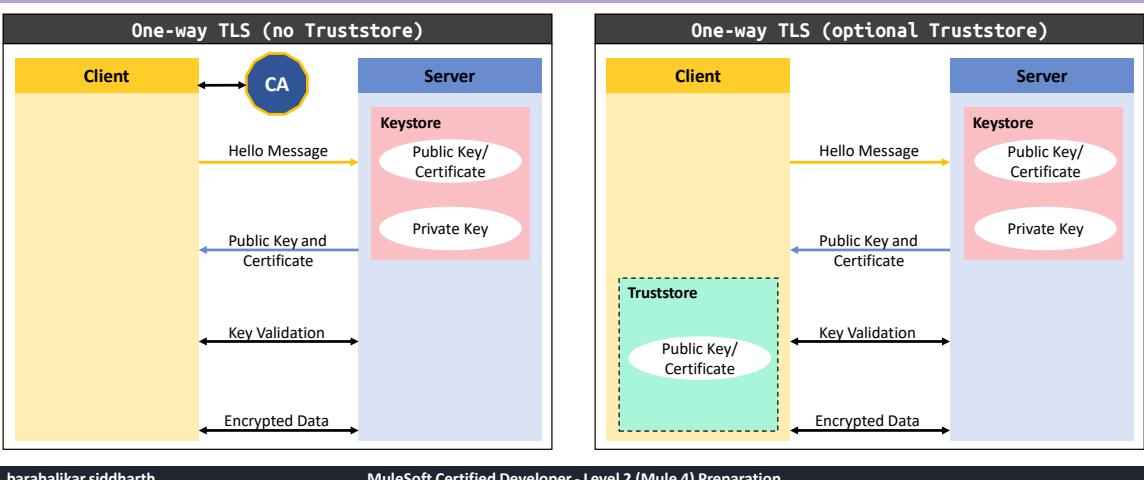
barahamercardonth

One-way TLS

MCD2

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 (<i>typically self-signed certificates</i>)

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



barahalikar.siddharth MuleSoft Certified Developer - Level 2 (Mule 4) Preparation

Lets understand what are keystores and truststores

Lets talk about one-way tls with no truststore

We have a client, a CA and a Server with keystore holding keys

- The client issues a session request to the server.
- The server responds with a certificate, which contains its public key.
- This certificate comes from the server's keystore, which also contains the server's private key.
 - The private key is never sent to the client.
- For a signed cert, the client makes a request to the Certificate Authority (CA) to authenticate the certificate.
- If certificate is valid
- The client and server exchange several more messages to validate keys.
- Then the client begins TLS data transfer with the server.

This is an example where client doesn't use a truststore, it validate certs with the CA

Now Lets talk about one-way tls with truststore

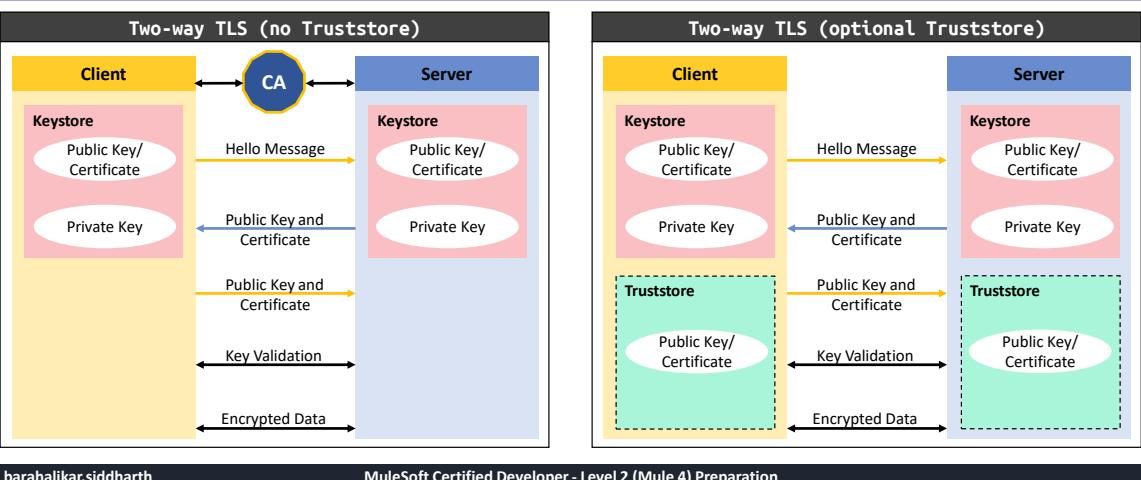
- **TRUSTSTORE**
- **It is the same as before but,**
- If the TLS server uses a self-signed certificate or a certificate that is not signed by a trusted CA, then you create a truststore on the client.
- The client populates its truststore with server certificates and public keys that it trusts.
- When the client receives a certificate, the incoming certificate is then validated against the certificates in its truststore.

Two-way TLS

MCD2

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 (<i>typically self-signed certificates</i>)

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



barahalikar.siddharth

MuleSoft Certified Developer - Level 2 (Mule 4) Preparation

Let talk about 2-way-tls following the same example discussed in previous slide

Here both the client and server have there own keystores.

The client issues a session request to the server.

The server and the client both responds with a certificate, which contains its public key.

For a signed cert, the client and server makes a request to the Certificate Authority (CA) to authenticate the individual certificates.

If everything is valid

The client and server exchange several more messages to validate keys.

The client begins TLS data transfer with the server.

Now Lets talk about two-way tls with optional truststore

- **TRUSTSTORE**
- **It is the same as before but,**
- If the Client & server uses a self-signed certificate or a certificate that is not signed by a trusted CA, then you create a truststore on both the client & server.
- The client populates its truststore with server public keys and server populates

- its truststore with client public key.
- When the client & server share certificates, the incoming certificates are then validated against the certificates in their truststore.

Architecture

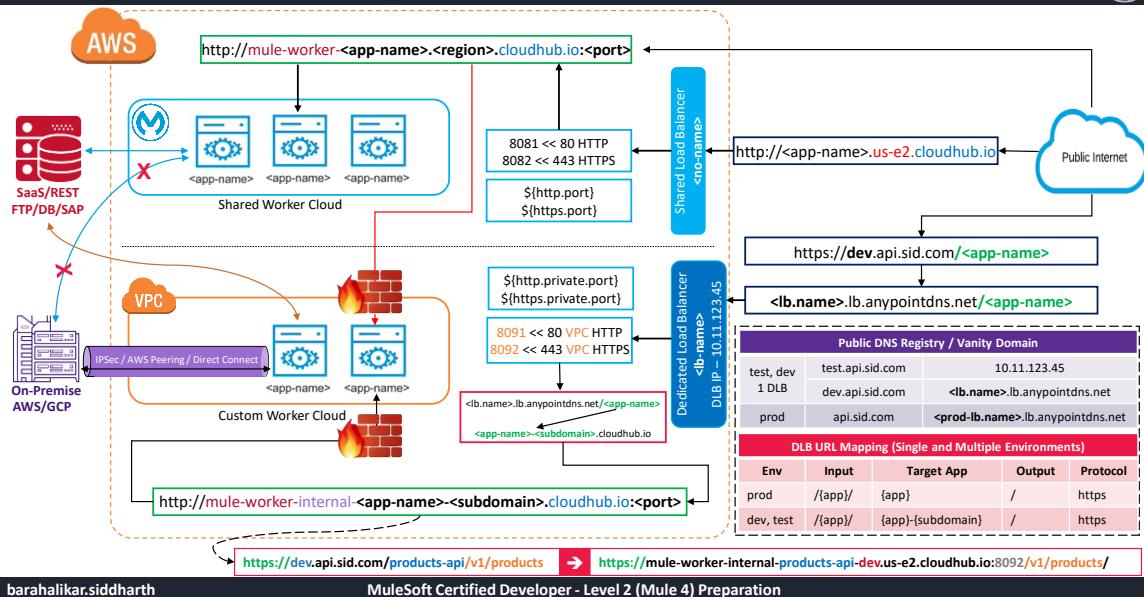
- CloudHub Architecture



barahamkarandhark
barahamkarandhark

CloudHub Architecture

MCD2



In your Mule app code, you use `api.port` instead of the more conventional `http.port`, `https.port`, `http.private.port`, or `https.private.port` for the name of the configuration property that holds the port (8081, 8082, 8091, or 8092, respectively) on which the API is exposed. This is so that an API implementation can, for example, move from the public CloudHub worker cloud to a VPC by just changing the value of `api.port` from 8082 to 8092 in a properties file, while keeping all Mule flow config files unchanged in using the property name `api.port`.

Self-Signed Cert

- Generate a Self-signed certificate
 - JKS vs PKCS12



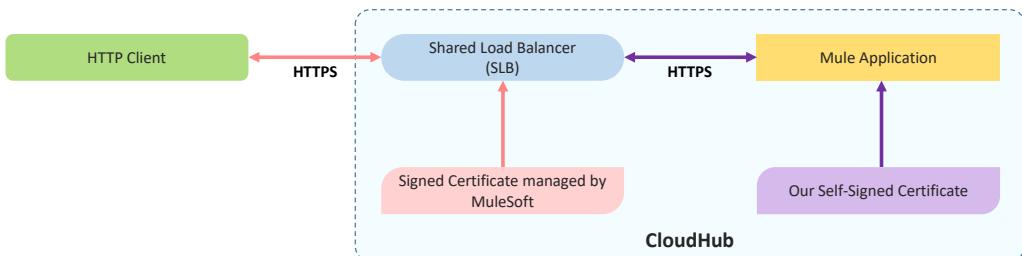
barahamkaranthanth

Create Self Signed Certificate – One-Way-TLS

MCD2

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
```

barahalikar.siddharth

MuleSoft Certified Developer - Level 2 (Mule 4) Preparation

In one way tls all we need is a pair of public private key stored in a keystore. We can generate them in multiple ways, one of them is using KEYTOOL cmd.

The default format used for these files is **JKS until Java 8**.

Since Java 9, **the default keystore format is PKCS12**.

The biggest difference between JKS and PKCS12 is that JKS is a format specific to Java, while PKCS12 is a standardized and language-neutral way of storing encrypted private keys and certificates.

We can use any of them based on the preference, I have mentioned both the cmds overs here for you reference.

Within Cloudbus, if we deploy a HTTPS based application behind a Shared Load balancer, then SLB is going to use it's own certificate to terminate TLS.

In cases where we want to terminate TLS using our certifications, we must deploy the application behind a Dedicated LoadBalancer

1.6 Self-Signed Cert & CloudHub

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



barahamercardorff

MCD2

Configuration Properties

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



barahmierandmehr

Configuration Values in Property Files

MCD2

<code>properties.yaml</code>	<code>properties-dev.yaml</code>	<code>properties-prod.yaml</code>	<code>properties-dev-secure.yaml</code>
<pre>1 api: 2 groupId: '\${api.groupId}' 3 artifactId: '\${api.artifactId}' 4 version: '\${api.version}' 5 majorVersion: 'v1' 6 port: '8082' 7 spec: 'resource:\${api.groupId}:\${api.id} \${api.id}: 7777777'</pre>	<pre>1 tls.keystore: 2 type: 'pkcs12' 3 path: 'certs/super-ticket-papi.p12' 4 alias: 'server' 5 6 api: 7 id: '1111111'</pre>	<pre>1 tls.keystore: 2 type: 'pkcs12' 3 path: 'certs/prod-super-ticket-papi.p12' 4 alias: 'server' 5 6 api: 7 id: '4444444'</pre>	<pre>1 tls.keystore: 2 keyPassword: '[[GldtU/4F1GrYHnFdmd6Yb8AgPFuSp]' 3 password: '[[GldtU/4F1GrYHnFdmd6Yb8AgPFuSp]'</pre>
File Structure	global.xml >> TLS Configuration	global.xml >> Property Configurations	
<pre>src/main/resources application-types.xml log4j2.xml properties.yaml properties-dev.yaml properties-dev-secure.yaml properties-prod.yaml properties-prod-secure.yaml</pre>	<pre>33<tls:context name="apiTLSContext"> 34 <tls:key-store 35 type="pkcs12" 36 path="\${api.keystore.path}" 37 alias="\${tls.keystore.alias}" 38 keyPassword="\${\${secure::tls.keystore.keypassword}}" 39 password="\${\${secure::tls.keystore.password}}"/></pre>	<pre>50 <configuration-properties file="properties-\${env}.yaml" /> 51 52<secure-properties:config file="properties-\${env}-secure.yaml" key="\${secure.key}"> 53 <secure-properties:encrypt algorithm="Blowfish" /> 54 </secure-properties:config> 55 56 <configuration-properties file="properties.yaml" /></pre>	
	Mule 4 – Wrong Order of Definition	Mule 4 – Correct Order of Definition	
	<pre>50 <configuration-properties file="properties.yaml" /> 51 52<secure-properties:config file="properties-\${env}-secure.yaml" key="\${secure.key}"> 53 <secure-properties:encrypt algorithm="Blowfish" /> 54 </secure-properties:config> 55 56 <configuration-properties file="properties-\${env}.yaml" /></pre>	<pre>50 <configuration-properties file="properties-\${env}.yaml" /> 51 52<secure-properties:config file="properties-\${env}-secure.yaml" key="\${secure.key}"> 53 <secure-properties:encrypt algorithm="Blowfish" /> 54 </secure-properties:config> 55 56 <configuration-properties file="properties.yaml" /></pre>	
	<pre>if \${env} = dev then \${api.id} = 7777777</pre>	<pre>if \${env} = dev then \${api.id} = 1111111</pre>	
barahalikar.siddharth	MuleSoft Certified Developer - Level 2 (Mule 4) Preparation		

It is best practice to centralize the management of properties rather than hard code values at the places. This allows for the reuse of values and better maintenance. Instead of changing the property in multiple places you change the value once in the property file.

Applications typically use different values in different environments
We develop a application and deploy or promote the same artifact to different environments

Whenever we change environment, it is not acceptable to make changes in artefacts.

So we make use of multiple property files each based on a specific environment. If you see here both dev and prod env use different API Instance ID.

Apart from these we also need to encrypt certain sensitive properties like passwords. We use separate property file for them as well. We will see how to encrypt them in a different session.

How do we use these property files?
We use placeholders in the connectors,

To access plain text value we use \${} and

The secure:: prefix is needed to access encrypted values from secure property files

The secure:: prefix is mandatory for all properties in a secure property file, even if one of the property is plain text

We have not used any plain text in our secure properties yaml, but if we had a plain text property, it should have been access using secure:: prefix. To avoid this we are using separate files for plain text and encrypted properties.

Within the global xml we add the CONFIGURATION PROPERTIES, but we have multiple files, so in which order should we define them?

Within Mule 4, it follows TOP 2 DOWN approach

The first property processed is PRESERVED

Later definitions cannot OVERRIDE them

This is a complete opposite behaviour compared to MULE 3

So in this example...

1.7 Configuration Properties

- Add properties.yaml



barahamkarandikar

MCD2

1.8 Configuration Properties

- Add environment specific property files
 - properties-dev.yaml
 - properties-prod.yaml
- Promote APIs to PROD env
 - Deploy to PROD

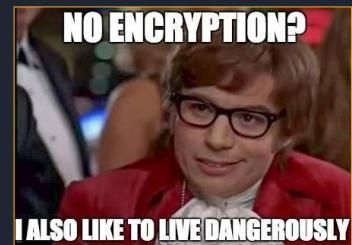


barahamkaranthath

MCD2

Secure & Safe Properties

- Encrypting properties
- Safe-Hidden properties



barahamercardonth

Secure and Safe Properties

MCD2

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: '!QMdtU/4F1GrYHnFGdM'
  password: '!QMdtU/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 thorough a Maven cmd.
```

```
mvn deploy -DmuleDeploy \
-M-Dsecure.key=dontLeakThisKeySecureKey@7 \
-M-Danypoint.platform.client_id=4janFGJu.. \
-M-Danypoint.platform.client_id=C9ONJgu2...
```

Runtime	Properties
<input type="button" value="Text"/>	<input type="button" value="List"/>
anypoint.platform.client_id=***** secure.key***** env=test anypoint.platform.client_secret=*****	

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

barahalikar.siddharth

MuleSoft Certified Developer - Level 2 (Mule 4) Preparation

<https://docs.mulesoft.com/downloads/mule-runtime/4.2/secure-properties-tool.jar>

It is critical to ensure that the sensitive information that a application stores is secure, protected from unauthorized users and malicious attackers. You can encrypt configuration properties as a security level for your applications.

We use the Secure Properties Tool to encrypt or decrypt text strings, values inside a properties file, or all the contents of a properties file.

Mulesoft provide a the tool as a JAR File, we download it and encrypt properties.

*****Example the command*****

This example shows a Admin/Ops teams encryption a single Text String using CBC mode of Blowfish algorithm, the value is encrypted and decrypted using a SECRETKEY. Details like the algorithm, mode used and encrypted value are sent to the Developers

Developers configure this value in a secure properties file

And make use of a Secure properties extensions module to decrypt this value during runtime

Since the developers are unaware of the secret key use, an dynamic placeholder is used in the place.

Finally the secret key is added to a secure-properties field in mule-artefact.json to hide the value in Clouduhub ui.

Developer packages the application and pushes it to a repo

The Admin/Ops or Cicd pipeline picks the app from repo and deploys it to Clouduhub by passing the secretKey as a property.

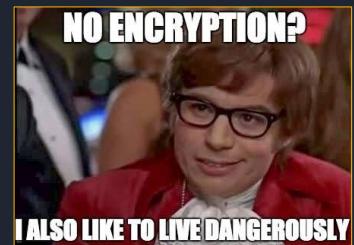
This key is used by secure properties extension to decrypt the values during runtime

And since we have used secure properties in Mule-artefact json, the secret key is masked in the Clouduhub UI >> this process is called as Hidden Properties.

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



barahamercardonth

(MCD2)

Maven

- Maven Basics



barahmierandmirth

Maven

MCD2

```

Maven Coordinates
groupId + artifactId + version
12<project ...
13  <groupId>53a7ba06-a8d3-4536-9fb5-5c689add373f</groupId>
14  <artifactId>super-tickets-papi</artifactId>
15  <version>1.0.0</version>
16 ...

```

```

24<dependency>
25  <groupId>org.mule.connectors</groupId>
26  <artifactId>mule-http-connector</artifactId>
27  <version>1.6.0</version>
28  <classifier>mule-plugin</classifier>
29 </dependency>

```

```

18<plugin>
19  <groupId>org.mule.tools.maven</groupId>
20  <artifactId>mule-maven-plugin</artifactId>
21  <version>3.5.2</version>
22  <extensions>true</extensions>
23 </plugin>

```

Lifecycle
clean
default
site

Phases
validate
initialize
compile
test
package
verify
install
deploy

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

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

Mule Maven Plugin

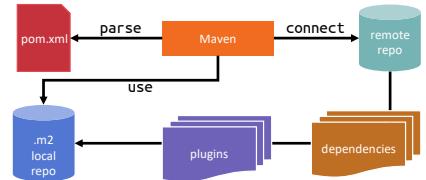
```

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

```

barahalikar.siddharth

MuleSoft Certified Developer - Level 2 (Mule 4) Preparation



```

settings.xml (Maven)
3 <settings>
4  <servers>
5   <server>
6     <id>anypoint-exchange-v3</id>
7     <username>Client</username>
8     <password>c8e4e6e79?f9f10E41f</password>
9   </server>
10 </servers>
11 </settings>

```

- We use Maven to avoid writing scripts for building projects and manually downloading the dependencies.
- Dependencies are nothing but libraries or JAR files.
- We need to specify required dependencies within a POM XML file.
- Maven parses the pom xml and checks for the dependencies within the local maven repo,
 - If it doesn't find them, it tries to get them from the central/remote repo
 - And stores them in the local maven repo

How does it find them?

- Maven uses coordinates a combination of groupId artifactId and version, for everything like project, dependencies and plugins defined in the pom xml
- For external credentials, it makes use of Mvn settings .xml to connect with external repos like anypoint exchange or mule maven enterprise repo

There is a specific life cycle that Maven follows to deploy and distribute the target project.

There are three built-in life cycles:

default – This is the main life cycle of Maven as it is responsible for project deployment.

clean – This life cycle is used to clean the project and remove all files generated by the previous build.

site – The aim of this life cycle is to create the project's site documentation

Each life cycle is made up of a sequence of phases.

A Maven phase is nothing but a stage in the Maven build life cycle. Each phase executes a specific task.

Here are a few important phases in the default build life cycle

validate – phase checks if all information necessary for the build is available

compile – phase compiles the source code

test-compile – phase compiles the test source code

test – phase runs unit tests

package – phase packages compiled source code into the distributable format (jar, war)

integration-test – phase processes and deploys the package if needed to run integration tests

install – phase installs the package to a local repository

deploy – phase copies the package to the remote repository

if you run the command mvn deploy, that is, the deploy phase which is the last phase in the default build life cycle, then this will execute all phases prior to the deploy phase as well.

Each phase is a sequence of goals, and each goal is responsible for a specific task.

When we run a phase – all goals bound to this phase are executed in order.

The syntax used is plugin:goal

Here are some of the phases and default goals bound to them:

The Mule Maven plugin enables you to integrate the packaging and deployment of your Mule applications with your Maven lifecycle.

Resource Filtering

- What?
- Why?
- How?



barahimkarandamir

Removing Configuration Redundancy

MCD2

```

properties.yaml
1@ api:
2   groupId: '53a7ba06-a8d3-4536-9fb5-5c689add373f'
3   artifactId: 'super-tickets-papi'
4   version: '1.0.0'

pom.xml
49@ <dependency>
50   <groupId>53a7ba06-a8d3-4536-9fb5-5c689add373f</groupId>
51   <artifactId>super-ticket-papi</artifactId>
52   <version>1.0.0</version>
53   <classifier>jas</classifier>
54   <type>zip</type>
55 </dependency>

pom.xml
19@ <properties>
20   <api.groupId>53a7ba06-a8d3-4536-9fb5-5c689add373f</api.groupId>
21   <api.artifactId>super-ticket-papi</api.artifactId>
22   <api.version>1.0.0</api.version>
23 </properties>

pom.xml
49@ <dependency>
50   <groupId>${api.groupId}</groupId>
51   <artifactId>${api.artifactId}</artifactId>
52   <version>${api.version}</version>
53   <classifier>jas</classifier>
54   <type>zip</type>
55 </dependency>

```

The diagram illustrates the process of removing redundancy. It shows four code snippets arranged in a grid. The top-left snippet is a properties file with a single entry. The top-right snippet is a pom.xml file that includes the properties file via the `<dependency>` tag. The bottom-left snippet is another pom.xml file that also includes the properties file via the `<dependency>` tag. Arrows point from the top-right and bottom-left snippets to the bottom-right snippet, which contains the Maven Resource Filtering configuration. The bottom-right snippet shows how the `filtering` parameter is used to replace placeholder variables like `api.groupId` and `api.artifactId` in the resources section.

```

properties.yaml
1@ api:
2   groupId: '${api.groupId}'
3   artifactId: '${api.artifactId}'
4   version: '${api.version}'

pom.xml
68@ <resources>
69@   <resource>
70     <directory>src/main/resources</directory>
71     <filtering>true</filtering>
72   </resource>
73 </resources>
74@ <testResources>
75@   <testResource>
76     <directory>src/test/resources</directory>
77     <filtering>true</filtering>
78   </testResource>
79 </testResources>
80@ <plugins>
81@   <plugin>
82     <groupId>org.apache.maven.plugins</groupId>
83     <artifactId>maven-resources-plugin</artifactId>
84     <version>3.2.0</version>
85@     <configuration>
86@       <nonFilteredFileExtensions>
87         <nonFilteredFileExtension>p12</nonFilteredFileExtension>
88         <nonFilteredFileExtension>crt</nonFilteredFileExtension>
89         <nonFilteredFileExtension>pem</nonFilteredFileExtension>
90       </nonFilteredFileExtensions>
91     </configuration>
92   </plugin>
93 </plugins>

```

Maven Resource Filtering

barahalikar.siddharth MuleSoft Certified Developer - Level 2 (Mule 4) Preparation

To avoid any duplicate configuration values we have Externalized the properties to a property yaml files in an earlier session

But duplicates arise when working with dependency properties. As we see here, both pom xml and properties yaml file use the same maven coordinates and can rise issues in the future. For example if the version of the api spec is updated, it needs to be changed in 2 places and will raise errors if we miss updating in either of the files

To avoid this we can make use of Maven resources plugin.

Maven Resource Filtering uses placeholders in resource files and replace them with actual value during configuration

This plugin has multiple goals like,

resources – goal which copies resources that are part of the main source code

testResources – goal copy resources that are part of the test source code

In the given configuration, there's a parameter named *filtering* with the value of *true*.

The **filtering** parameter is used to replace placeholder variables in the resource files.

Apart from goals we can also add some configuration to filter some extensions which don't require filtering.

We can make use of properties within pom xml to use the values for filtering.

These values are referred in dependency using placeholders,
The same values can also be referred in properties yaml file

So in the future any changes to coordinates can be done in pom properties alone and
It will be referred in multiple places

In this way we can avoid duplicate values in multiple files

1.10 Resource Filtering

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



MCD2

barahimkarandamir

Build Redundancies & Duplication

- Dependency Management
- POM
 - Parent POM
 - Master BOM



barahmierandforth

Dependency & Plugin Management

MCD2

```
    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 ...

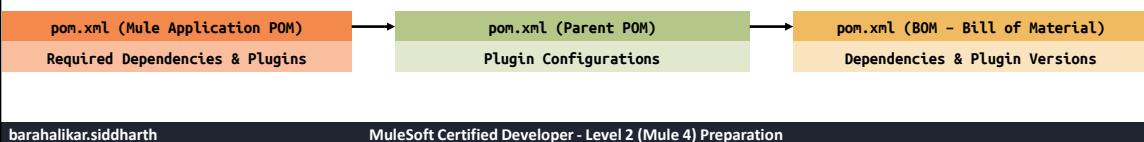
```

```
 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>XLARGE</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   ...

```



Lets understand why Dependency & Plugin Management is a complex task by taking a simple example,

Lets assume we have a Developer John working on app called Project_10.
His pom xml contains a mule maven plugin of version 1.1.1, using 1 worker of type MICRO and also has a http connector of version 1.1.1
His maven package command is failing and is unable to fix it,
So he connects with his teammate Sara

Sara is also a developer working on the same Project10 app, Upon discussion they found that Sara's mvn package command is working as expected, but they Found that they are using different configurations values. Sara's pom defined mule maven version as 2.2.2 and used 2 workers of type LARGE and use 1.2.2 versoon of http connector.

Now both of them are not sure which configurations is correct.

They reach out to another developer Sid working on a different app named

Project_15

Sid's pom configuration uses all latest versions of plugins and dependencies

This issue arises when a organization doesn't have a standard build configuration in place.

This can be avoid by extracting the MAVEN Plugin configurations that are not specific to a mule app to a reusable Parent Pom.

Dependencies management can be extracted into their own parent-pom called as BOM-bill of materials

The reason we have a separate pom for versions management and plugin management is because it is more common to update the dependency/plugin versions compared to plugin configurations

In this way we can reduce redundancy by making all project use a centralized parent-pom and bom. Any changes to these pom's will be reflected in all projects.

POM << Parent-POM << BOM

```

..../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 ...
170 ...

..../parent-pom/pom.xml (Parent POM)
1 <parent>
2   <groupId>53a7ba06-a8d3-4536-9fb5</groupId>
3   <artifactId>parent-pom</artifactId>
4   <version>1.0.0</version>
5   <relativePath>..../parent-pom/pom.xml</relativePath>
6 </parent>
7 ...
8 <properties>
9   <type>custom</type>
10  <deployment.prefix>sid-</deployment.prefix>
11  <deployment.suffix>-${deployment.env}</deployment.suffix>
12  <deployment.name>${deployment.prefix}...</deployment.name>
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>${deployment.name}</applicationName>
29           <properties>
30             <env>${deployment.env}</env>
31             </properties>
32           </cloudHubDeployment>
33         </configuration>
34       </plugin>
35     </plugins>
36   </pluginManagement>
37 ...
38 ...

..../bom/pom.xml (BOM)
1 ...
2 <properties>
3   <app.runtime-server>4.4.0</app.runtime-server>
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     <uri>https://maven.....com/api/v3/org/${org}/maven</uri>
33   </repository>
34 </distributionManagement>
35 <pluginRepositories>
36   <pluginRepository>
37     <id>mulesoft-releases</id>
38     <name>MuleSoft Releases Repository</name>
39     <layout>default</layout>
40     <url>https://repository.mulesoft.org/releases/</url>
41     <snapshots>
42       <enabled>false</enabled>
43     </snapshots>
44   </pluginRepository>
45 </pluginRepositories>

```

barahalikar.siddharth

MuleSoft Certified Developer - Level 2 (Mule 4) Preparation

Lets understand how this works in a Mule app,

Lets start with the BOM, which manages the versions of both dependencies and plugins

Within Parent-pom we define the plugin configuration for example the Mule maven configuration for Cloudhub deployment

Within Mule app we just define which dependency and plugins are required. If you notice we do not define any versions on mule app pom and parent pom.

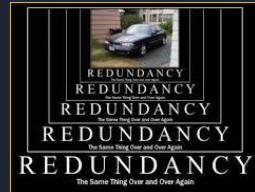
As all versions are managed by a BOM

And at the same time mule maven configuration is part of the parent-pom

Any new project can reuse the parent-pom and bom-pom configuration to avoid duplications and redundancies.zz

1.11 Parent-POM

- Configure Parent-POM

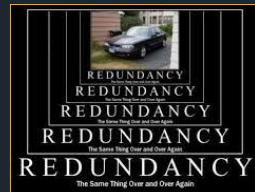


barahmaranth

(MCD2)

1.12 Master-BOM

- Configure Master-BOM



MCD2

barahim karandikar

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



barahatkarandhark
barahatkarandhark

(MCD2)

1.14 Connected App

- What?
- Why?
- How?
- Use credentials in settings.xml
- Build application



barahamcaronduh

MCD2

Mule Connected Apps

MCD2

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

The screenshot shows the Mule Connected Apps interface. On the left, there's a 'Create App' section where the name is set to 'CH-Deploy-App'. Under 'Type', the option 'App acts on its own behalf (client credentials)' is selected, highlighted with a red box. Below this, there's a 'Select scopes' section with several options like 'Cloudhub Organization Admin', 'Read Applications', etc., each with a toggle switch. To the right, there's a 'Scopes Required to Publish and Consume Artefacts from Exchange' table with columns for 'Scopes' and 'Business Groups'. The table lists 'Exchange', 'Exchange Contributor', 'Exchange Viewer', and 'Profile' under 'General'. On the far right, a code block shows '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>
```

barahalikar.siddharth

MuleSoft Certified Developer - Level 2 (Mule 4) Preparation

The Connected Apps feature provides a framework that enables an external application to integrate with Anypoint Platform using [APIs](#) through OAuth 2.0 and OpenID Connect.

Only an [organization administrator](#) can view and manage connected apps feature for the entire organization.

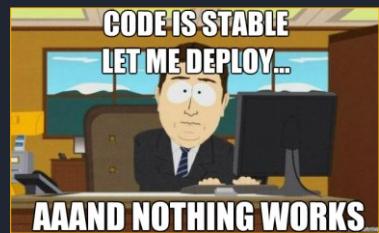
Developers can delegate access to applications that use APIs to interact with Anypoint Platform programmatically. For example while building Mule apps using CICD, the scripts needs appropriate credentials to deploy the application to Cloudbus, the traditional way is to embed users username and password within the maven settings xml.

This is not recommended because, the user credentials might be exposed and can be used for privileged anypoint platform access.

To avoid this we can make use of connected apps.

Maven Deployment

- CloudHub Deployment
- Connected App
 - Scopes



barahamkaranth

Maven CloudHub Deployment

MCD2

```
mvn -DmuleDeploy deploy \
-Dap_client_id=65f0afff0bc38bb1932d \
-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

```

35 <properties>
36   <deployment.prefix>sid-</deployment.prefix>
37   <deployment.suffix>${deployment.env}</deployment.suffix>
38   <deployment.name>${deployment.prefix}${project.artifactId}${deployment.suffix}</deployment.name>
39   <ap.environment>${deployment.env}</ap.environment>
40 
41 <plugin>
42   <groupId>org.mule.tools.maven</groupId>
43   <artifactId>mule-maven-plugin</artifactId>
44   <!-- <version>3.5.3</version> -->
45   <extensions>true</extensions>
46 </plugin>
47 <configuration>
48   <cloudHubDeployment>
49     <businessGroup> /</businessGroup>
50     <environment>${ap.environment}</environment>
51     <region>us-east-2</region>
52     <muleVersion>${app.runtime.semver}</muleVersion>
53     <applyLatestRuntimePatch>true</applyLatestRuntimePatch>
54     <workers>1</workers>
55     <workerType>MICRO</workerType>
56     <objectStoreV2>true</objectStoreV2>
57     <applicationName>${deployment.name}</applicationName>
58     <deploymentTimeout>600000</deploymentTimeout>
59     <connectedAppClientId>${ap.conapp.client_id}</connectedAppClientId>
60     <connectedAppClientSecret>${ap.conapp.client_secret}</connectedAppClientSecret>
61     <connectedAppGrantType>client_credentials</connectedAppGrantType>
62 
63 <properties>
64   <secure.key>${secure.key}</secure.key>
65   <env>${deployment.env}</env>
66   <anypoint.platform.client_id>${ap.client_id}</anypoint.platform.client_id>
67   <anypoint.platform.client_secret>${ap.client_secret}</anypoint.platform.client_secret>
68   <anypoint.platform.config.analytics.agent.enabled>true</anypoint.platform.config.analytics.agent.enabled>
69   <anypoint.platform.visualizer.displayName>${project.artifactId}</anypoint.platform.visualizer.displayName>
70   <anypoint.platform.visualizer.layer>${api.layer}</anypoint.platform.visualizer.layer>
71 
72 </properties>
73 </cloudHubDeployment>
74 </configuration>
75 </plugin>
```

barahalikar.siddharth

MuleSoft Certified Developer - Level 2 (Mule 4) Preparation

In addition to using Anypoint Studio, Anypoint Runtime Manager, or the Anypoint Platform CLI to deploy applications to CloudHub, you can also deploy, redeploy, or undeploy applications by using the Mule Maven plugin.

To do so, we must add the mule maven plugin and cloudhub deployment strategy in the project's pom.xml or parent-pom.

applyLatestRuntimePatch When set to true, the plugin instructs CloudHub to update the worker to the latest available patch for the Mule runtime engine version specified in the deployment configuration, and then deploys the application.

Authentication Methods

When you deploy applications using Mule Maven plugin, you can use different methods to provide your credentials to authenticate against the deployment platform. Depending on the authentication method you use, the parameters to set in the deployment configuration differ

Different mode .. Explain

Here I have use connected apps and hence configured them in the plugin

For deployment we can either use mvn deploy or pass in an argument –
DmuleDeploy

Both of them deploy the app to Cloudbus, but

muleDeploy

Instructs the plugin to deploy using the deployment strategy defined in the plugin configuration. If the muleDeploy parameter is not set, the plugin uploads the artifacts to the repository defined in the distributionManagement section of the application's pom.xml file.

we can use a simple mvn mule:undeploy command to undeploy app from cloudbus

<https://docs.mulesoft.com/mule-runtime/4.4/mmp-concept>

<https://docs.mulesoft.com/mule-runtime/4.4/mmp-concept>

Mule Connected Apps

MCD2

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

Create App

Name
CH-Deploy-App

Type
 App acts on behalf of a user
Authorized by a user to act on their behalf.
 App acts on its own behalf (client credentials)
Acts on its own behalf without impersonating a user. The app can only be used in this organization.

Select scopes

Cloudhub Organization Admin ⓘ	<input checked="" type="checkbox"/>
Read Applications ⓘ	<input checked="" type="checkbox"/>
Create Applications ⓘ	<input checked="" type="checkbox"/>
Delete Applications ⓘ	<input checked="" type="checkbox"/>

Business Groups Environments

Super-Org-Mento	<input checked="" type="checkbox"/>
training	<input type="checkbox"/>
dev	<input type="checkbox"/>
prod	<input checked="" type="checkbox"/>

Id
8580cdc4f84e4528848b708299610c7e

Secret
763065Ba81cF49bC9c4ff6e033d5A565

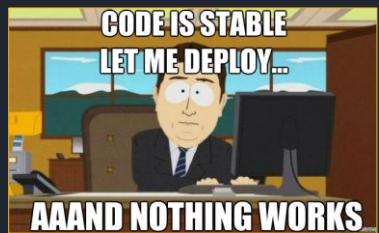
Scopes	Business Groups & Environments
Design Center	
Design Center Developer ⓘ	Super-Org-Mento <input type="button" value="Manage"/>
Exchange	
Exchange Administrator ⓘ	Super-Org-Mento <input type="button" value="Manage"/>
Runtime Manager	
Cloudhub Organization Admin ⓘ	Super-Org-Mento <input type="button" value="Manage"/>
Create Applications ⓘ	Super-Org-Mento (2 environments) <input type="button" value="Manage"/>
Delete Applications ⓘ	Super-Org-Mento (2 environments) <input type="button" value="Manage"/>
Download Applications ⓘ	Super-Org-Mento (2 environments) <input type="button" value="Manage"/>
Manage Application Data ⓘ	Super-Org-Mento (2 environments) <input type="button" value="Manage"/>
Read Applications ⓘ	Super-Org-Mento (2 environments) <input type="button" value="Manage"/>
General	
Profile ⓘ	<input type="button" value="Remove"/>

barahalikar.siddharth

MuleSoft Certified Developer - Level 2 (Mule 4) Preparation

1.15 Maven Deployment

- CloudHub Deployment
- Connected App
 - Scopes

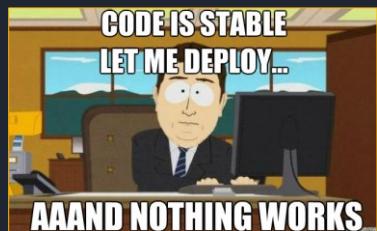


barahamercardorff

MCD2

1.16 Anypoint Visualizer

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



barahamkaranthanth

MCD2

Anypoint Visualizer

The screenshot shows the Anypoint Visualizer interface with the following components:

- Mule app pom.xml**: A code editor window showing XML configuration for a Mule application.
- Parent-pom pom.xml**: A code editor window showing XML configuration for a parent Mule application.
- Configuration Snippets**: A list of configuration snippets:
 - `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`
- API Details**: A panel for the API `papi-user-flights` showing details like Organization (hcl-ops-1), Environment (Sandbox), Type (Mule app), Runtime version (4.3.0), and Hostname (papi-user-flights.us-e2.cloudhub.io). It also lists `api-user-flights-pet` and `api-user-flights-pet` under the `Manage policies` section.
- Policy Details**: A panel for the API `papi-user-flights` showing `API level policies` (HTTP Caching, Rate limiting) and a link to `Select dependents and dependencies`.
- Architecture Graph**: A central graph showing the system architecture. Nodes include `External Traffic`, `api-user-flights-pet`, `papi-user-flights`, `papi-pets`, `api-users`, `api-myapp`, `mule-pets.firebaseio.com`, `api-myapp-api`, and `regress.in`. Arrows indicate flow between these components across layers: Experience, Process, System, Backend, and Database.

Anypoint Visualizer provides a real-time, graphical representation of the APIs, and Mule applications that are running and discoverable.

Anypoint Visualizer collects data from Mule applications and APIs to display these graphs.

Anypoint Visualizer has three use cases, or *visualizations*: Architecture, Troubleshooting, and Policies.

- The [Architecture visualization](#) shows the topology of your apps and APIs. This visualization is useful for understanding your system architecture.
- The [Troubleshooting visualization](#) enables you to view the health of the system and evaluate performance-based metrics.
- The [Policy visualization](#) enables you to see the policy landscape of your application network at a glance.

to Enable Visualization monitoring we need to add a property to each application deployed to CloudHub.

`anypoint.platform.config.analytics.agent.enabled=true`

To associate an application with a specific layer like Process , System, Experience, Backend a property name.....

anypoint.platform.visualizer.layer=Process

And to use a specific display name, we use

anypoint.platform.visualizer.displayName=<project-artifactId>

So where do we define these properties?

Section 1 - Completed

MCD2

#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

barahalikar.siddharth

MuleSoft Certified Developer - Level 2 (Mule 4) Preparation

Welcome to Section 1, in this section we will start by adding an Open API Specification to Design center,

Configure API instance for it in API manager and Import it to Studio

Within Studio, we will add best coding conventions and setup the project structure. We will Learn about Cryptogrpahy and how to use it for secureing communicatons/data in transit.

For static data we will use Secure and Hidden properties to encrpt and mask them

We will also understand Cloudhub architecture and setup a Maven based deployment strategy.

We will also modularize mule apps using parent-pom and bill of materials concepts. We will end this section by visualizing the API interactions within 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:

1 2 3

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

barahalikar.siddharth

MuleSoft Certified Developer - Level 2 (Mule 4) Preparation

This course doesn't talk about what mulesoft is or what it is used for, because if you are preparing for this Level 2 certification, you already know about it.

As we know, there are 3 major Level 1 Mule certifications, which are MCD L1, MCIA L1, MCPA L1

This course is specifically designed for MCD L2 certification which is all about working independently on production ready mule apps.

It has 5 major topics like

Expose production ready anypoiny platform managed apis from Mule applications

Implement monitorable mule apps

Secure data at rest and in transit

And others

These 5 topics have around 30+ subtopics

We will dive into each of these topics, by first understand the concept and then implementing it

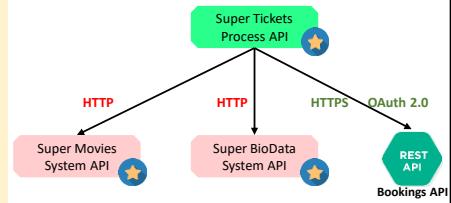
So how do we do this?

Section 2

MCD2

#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



barahalikar.siddharth

MuleSoft Certified Developer - Level 2 (Mule 4) Preparation

- Welcome to Section2, in this section we will start adding the integration logic to our Super tickets Papi
- We will add system api rest connectors in studio
- Get credentials, apply non functional requirement and invoke SAPI
- We will discuss about Oauth in detail and configure http request connector to call an oauth service to get tokens and use the token in header
- Watch Oauth server interactions using a DEBUG Logger
- To improve performance we will use Scatter gather for parallel execution of these apis, followed by
- Handling errors for scatter gather and the Mule application following the concepts of transient and permanent http errors
- We will end this section by implementation both Client side and server side caching.

REST Connect

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



HTTP Request
Connector

REST
Connect

barahimkarandam



REST Connect quickly converts an API specification (RAML 1.0, OAS2 or OAS3) to a connector,

How is it done?

A developer writes API Specification in design center and publishes it to exchange
The Exchange backend uses REST Connect to transparently convert a REST API specification to a Mule 3 and Mule 4 connector.

REST Connect DOES NOT currently support custom TLS configurations

But supports other security schemas like basic auth, OAuth and others

It can be downloaded or used as a connector in Anypoint Studio.

To add a custom Display name to each resource, we can use display name field or a REST Connect Library

This is the code for the library and if you remember we have used this in one of our earlier RAML

Points to be noted,

REST Connect generates metadata for each operation based on your

schema definition in the request and response for each method in your RAML.

REST Connect cannot generate metadata based on examples in the RAML.

You need to define a schema to generate metadata.

2.1 REST Connect

- How to configure in Mule app?



HTTP Request
Connector

REST
Connect

MCD2

barahimcarthorh

2.2 REST Connect

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



HTTP Request
Connector

REST
Connect

barahimkarandam

MCD2

2.3 REST Connect

- Add timeouts to follow
 - Non-functional requirement



MCD2

API requirements include functional & non-functional requirements

A functional requirement defines (what your API should do)

example of functional requirement are like,
the api should be accessible in a mobile application.
api should be Servicing financial transactions.
api should provide a self-serve portal.

Where as nonfunctional requirements define (how your API should perform in terms of service level agreements).

examples of non functional requirement could be service Availability, scalability, security policies, response timeouts , latencies and etc

Non-functional requirements are essential **to ensure the usability and effectiveness of the entire system**

In this session we will look into response timeouts

OAuth 2.0

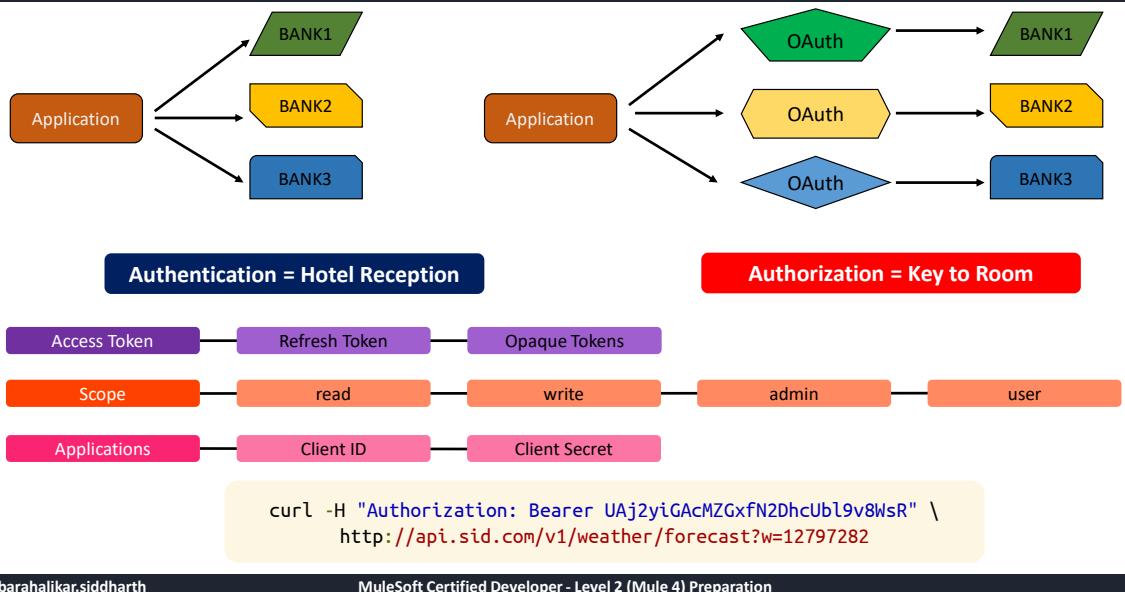
- What/Why/How?
- Authorization Code Grant type steps



barahimkarandamir

OAuth – Open Authorization

MCD2

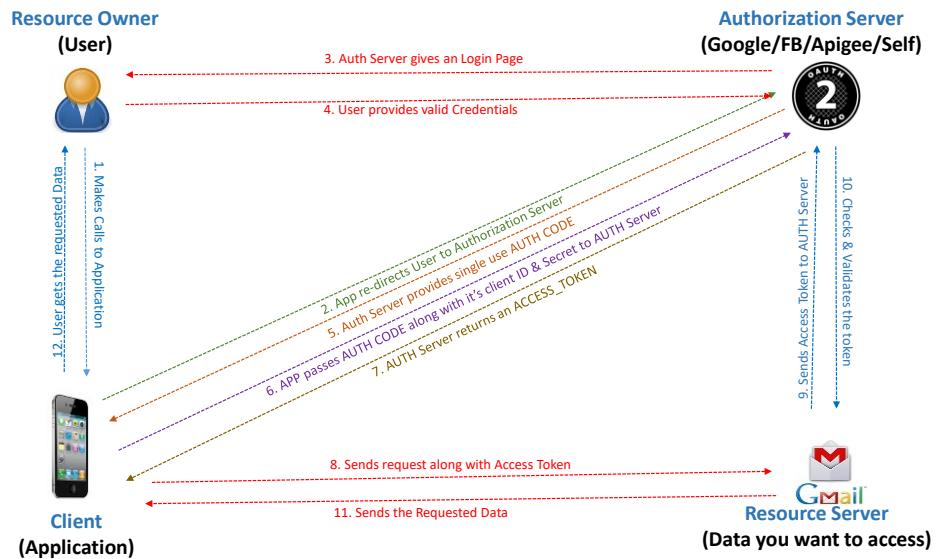


The OAuth (open authorization) protocol was developed to **enable secure delegated access**. It lets an application access a resource that is controlled by someone else.

Lets assume we have 3 bank accounts

OAuth – Authorization Code Grant Type Flow

MCD2



barahalikar.siddharth

MuleSoft Certified Developer - Level 2 (Mule 4) Preparation

If PKCE is too be used..

Step 2 – Client App re-directs User to Authorization Server along with an **Code_verifier + Code_challenge**

Step 6 – Client makes a call with **Authorization_Code + Code_verifier**

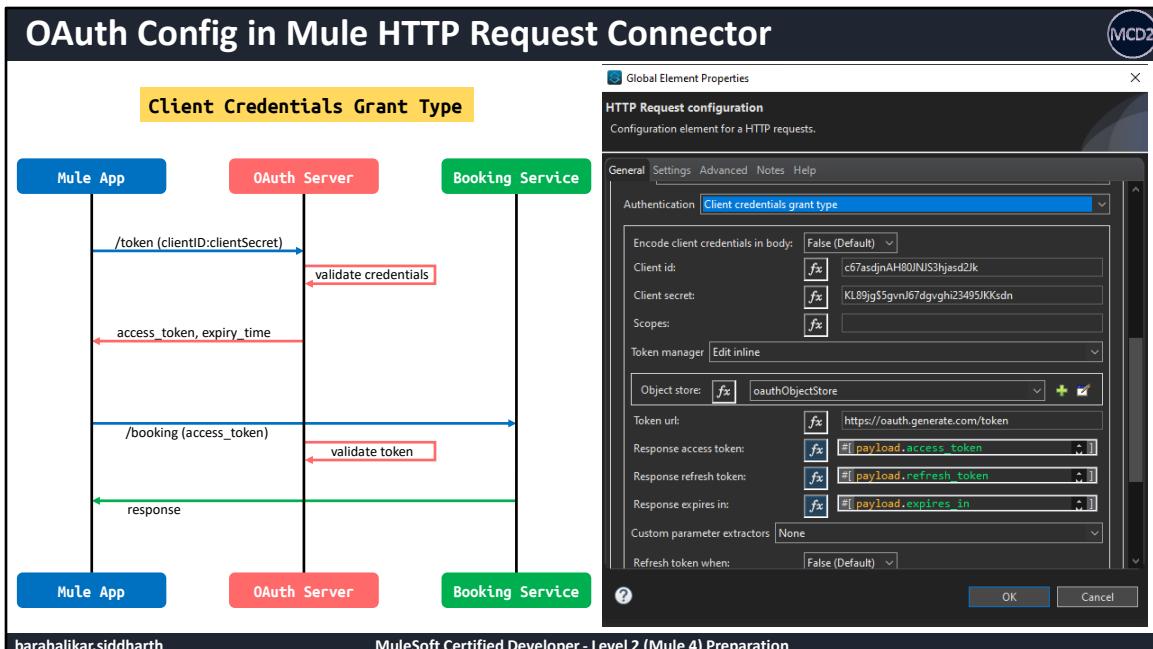
OAuth 2.0

- Configure OAuth in HTTP Request Connector



MCD2

barahamercardonth



HTTP Connector) **Request** operation supports connecting a Mule client app to a service that requires any of the following types of authentication,

[Basic Authentication](#)

[Digest Authentication](#)

[NTLM Authentication](#)

[OAuth2 Authorization Code Grant Type Authentication](#)

[OAuth2 Client Credentials Grant Type Authentication](#)

Mule runtime engine (Mule) uses the credentials configured in the HTTP request connector to configure an authorization header in the HTTP request.

In this example I have configures OAATH client credentials details in the http connector and this is how the flow works

We have mule app and a booking service secured by a oauth server

- Mule app make a calls to oauth server requesting a token by using the configuration defined in the authentication tab
- The oauth server validates the credentiais and if everything is correct, it generates a token, with some expiry time and send it back to Mule app

- Now mule app will use this token as a authorization header and make a call to the actual booking service,
 - The access token is validated by the booking service using the oauth server and if the token is valid
 - The response is sent back
-
- SO when a client makes a call to the booking service, the Mule engine runs through all these steps to get the response.

2.4 OAuth 2.0

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



babaramezandarani

MCD2

2.5 OAuth 2.0

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



barahimkarandamir

MCD2

2.6 Response Modification

- Add transformation logic



barahamercardonth

MCD2

2.7 Error Handling

- Handling custom errors

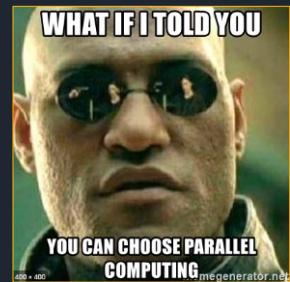


MCD2

barahmierandmorth

Parallel Processing

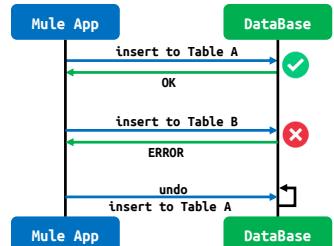
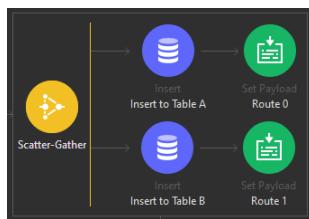
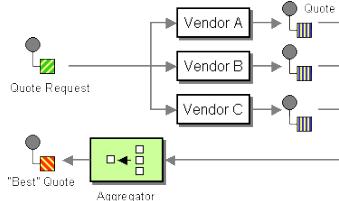
- Use Scatter-Gather
- Error Handling



barahamercardonth

Scatter Gather

MCD2



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
  
```

barahalikar.siddharth

MuleSoft Certified Developer - Level 2 (Mule 4) Preparation

As per enterprise integration pattern a **Scatter-Gather** broadcasts a message to multiple recipients and re-aggregates the responses back into a single message.

The Mule Scatter-Gather component is a **routing event processor that processes a Mule event through different parallel processing routes that contain different event processors.**

Within a scatter gather, if `maxConcurrency = 1`, then Scatter-gather processes the routes **sequentially**

Let talk abou this example,

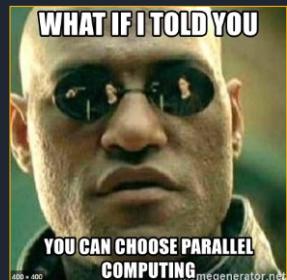
- We have a mule app with scatter gather making parallel calls to two MYSQL tables A and B
- It is going to insert to both tables in parallel and wait for the reponse.
- Lets assume insert to table b failed with an error
 - If a route fails, it raises MULE:COMPOSITE ERROR
 - If it is a timeout, it raises MULE:TIMEOUT
- In this case I want to undo insert to the success route which is TABLE A
- How do we achieve this?

- You may be thinking of putting scatter gather inside a TRY scope with some TRANSACTION BEGIN or JOIN option.
- That may work but, when you run scatter gather within a transaction, scatter gather does not execute in parallel.
 - Well this defeats the whole purpose of using scatter gather in the first place?
- So how do we undo/compensate success route?
- We can make use of the errorMessage find out the successroute and apply some compensaton logic to it
- When we get a mule:composite error, within the errorMessage we get a payload of failures and results, which has the info on the routes
- This info can be used to apply a compensation logic

There is lots more to learn about Scatter gather, I would recommend you to go to Mule documentation and scan the entire page.

2.8 Parallel Processing

- Use Scatter-Gather

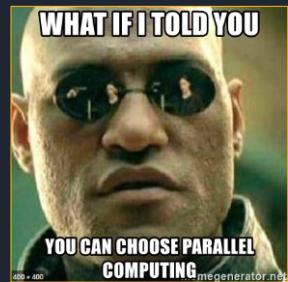


barahamkarandanjani

MCD2

2.9 Parallel Processing

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



barahamkarandanjani

MCD2

2.10 Parallel Processing

- Use Scatter-Gather
 - Compensation Logic



barahimkarandjohh

MCD2

Errors and Retry

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

PUT IT ON REPEAT



barahimkarandjith

Errors and Retry

MCD2

2xx Success		3xx Redirection		4xx Client Error		5xx Server Error					
200	Success	301	Permanent Redirect	401	Unauthorized Error	501	Not Implemented				
201	Created	302	Temporary Redirect	403	Forbidden	502	Bad Gateway				
204	No-Content	304	Not Modified	405	Method Not Allowed	503	Service Unavailable				
				429 Too Many Requests		504 Gateway Timeout					
Transient Errors				Permanent Errors							
429 503 504				401 403 405 501							
Reconnection Strategy Connection - Reconnection strategy: Standard Frequency (ms): 2000 Reconnection Attempts: 2		HTTP Request Connector Response - Response validator: Success status code validator Values: fx 200..399		Until Successful Scope Display Name: Until Successful Generic - Max Retries: fx 5 Milliseconds Between Retries: fx 60000							
barahalikar.siddharth		MuleSoft Certified Developer - Level 2 (Mule 4) Preparation									

HTTP response status codes are three-digit responses that indicate whether a specific [HTTP](#) request has been successfully completed. Responses are grouped in five classes:

- [Informational responses](#) (100–199)
- [Successful responses](#) (200–299)
- [Redirection messages](#) (300–399)
- [Client error responses](#) (400–499)
- [Server error responses](#) (500–599)

There are about 40 different status codes and we will cover the most widely used ones.

Anything from 2xx to 3xx are deemed to be a success.

4xx and 5xx are request breaking errors and we need to understand which of them are transient errors and what are permanent errors.

Transient errors are temporary and usually go away rapidly. This is why using a Retry Policy to encapsulate code is recommended.

Some examples are 429 too many requests, 503 Service Unavailable and 504 Gateway Timed out

Permanent error will not be resolved even after multiple retries so we raise an error straight way for those status codes

Some examples are 401 403 405 501

In Mule apps for transient error, we could retry the request using options like Reconnection strategy and Until Successful scope

A Reconnection Strategy can Set the number of reconnection attempts and the interval at which to execute them before returning a connectivity error

A Until Successful scope processes the components within it, until they succeed or exhaust the maximum number of retries.

Within Http request connector we can also configure what responses code considered as successful using the response validator option, we will discuss more on this in upcoming sessions

2.11 Errors and Retry

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

PUT IT ON REPEAT



barahimkarandam

MCD2

Caching

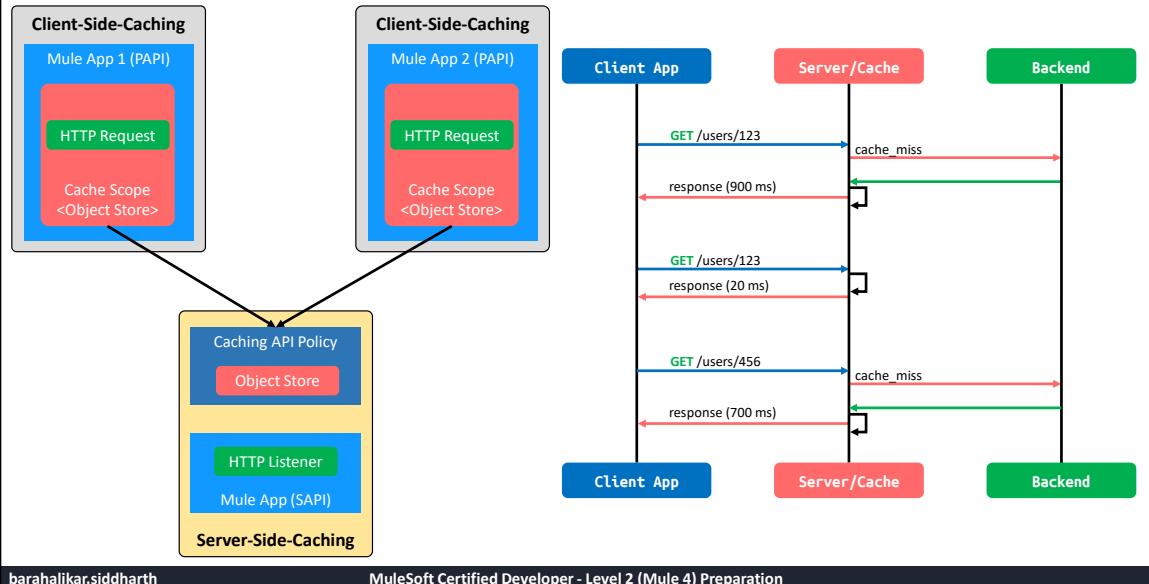
- What is caching?
- Type of caching
 - Server-side
 - Client-side
- Cache Scope
- Object Store



barahimkarandam

Caching

MCD2



Caching is the process of storing copies of files in a **cache**, or temporary storage location, so that they can be accessed more quickly. We can also configure TTL for each cache entry. Once expired the cache entry will be deleted.

It's typically only done for HTTP GET requests to **improve speed**, because we want to deliver fast content to our consumer and also to **reduce server load**, because we don't want our servers to compute without the need of it

Lets understand how caching works,

- We have a client app, and a backend system which uses a Caching server
- When the client make a **GET /users/123** request, the cache server will forward the request to the backend, because this is the very first call and there is nothing there in the cache. It is often denoted by **cache_miss**
- When the backend responds, a copy of the response is stored on the cache server and the response is sent back to the client
- Notice the response time of 900ms.

When client makes a 2nd call for the same user, now the response will be sent back from the cache server and hence the response time is reduced.

Here the call is never sent/processed by the backend and hence reducing its load

When a client makes a 3rd call for a different user, cache will be missed and call will be forwarded to the backend.

This is how a typical caching works

We have 2 types of cache, lets try to understand them using Mule terminology

Lets assume we have a System API which listen on a HTTP Listener.

We can use a Caching API Policy in front of it to reduce the SAPI processing load.

This is known as Server side caching

Lets assume we have 2 more Process API with HTTP Request connectors making external calls to the SAPI.

To improve the performance of these application and to improve speed, we can encapsulate the HTTP request connectors inside a CACHE SCOPE.

This is known as Client side caching

Both cache scope and caching api policy uses object store to persist data.

Object Store

MCD2

	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

sid-super-tickets-papi

Object Stores	Partitions	Keys	Values
Name	Partition	Key	Value Type
<input checked="" type="checkbox"/> defaultPersistentObjectStore	<input checked="" type="checkbox"/> biodataObjectStore	<input checked="" type="checkbox"/> MARVEL0002	<input type="checkbox"/> [binary value] BINARY
<input type="checkbox"/> httpCachingPolicyObjectStore_2603889	<input type="checkbox"/> oauthTokenObjectStore		

Viewing Object Store Data

Global Element Properties

Object store
Global Object store configuration

General Notes Help

Basic Settings

Name:

Object store

Persistent

Max entries:

Entry ttl:

Expiration interval:

Expiration interval unit:

Configuration Reference:

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

barahalikar.siddharth
MuleSoft Certified Developer - Level 2 (Mule 4) Preparation

<https://docs.mulesoft.com/object-store-connector/1.2/object-store-to-define-a-new-os>

Object Store Connector is a Mule component that allows for simple key-value storage. Although it can serve a wide variety of use cases, it is mainly design for:
 Storing information such as access tokens.
 Storing user information.

Additionally, Mule Runtime uses Object Stores for components, like:
 The Cache module uses an Object Store to cached data.
 The OAuth module (uses Object Stores to store the tokens).

There are three types of Object Store are:

Anypoint Object Store V2
 CloudHub Object Store V1
 Mule Object Store

Anypoint Object Store V2

is the latest version of CloudHub Object Store. This is a Cloud Service that is external to the Mule Application. It is only used by CloudHub deployed applications.
We can view the object store data in Cloudhub ui

CloudHub Object Store V1 (OSv1)

is the original Cloud-based Object Store for CloudHub applications. This version is now deprecated

Mule Object Store

is the original on-premise based Object Store that is part of Mule Runtime. this one is fully customisable to user preferences,

Please check out key information regarding each object store

The same Mule Object Store Connector can be used for all three object stores and is the recommended method to write and read key:values to Object Stores.

A custom object store can be configured using Object Store Global elements,
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**

in this image we have set the entry ttl to 1 hour and expiration interval to 30 minutes,
So

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

Again I would recommend going through the mule documentation to know about object stores

2.12 Client-side Cache

- Cache Scope
- Object Store



barahmierandmih

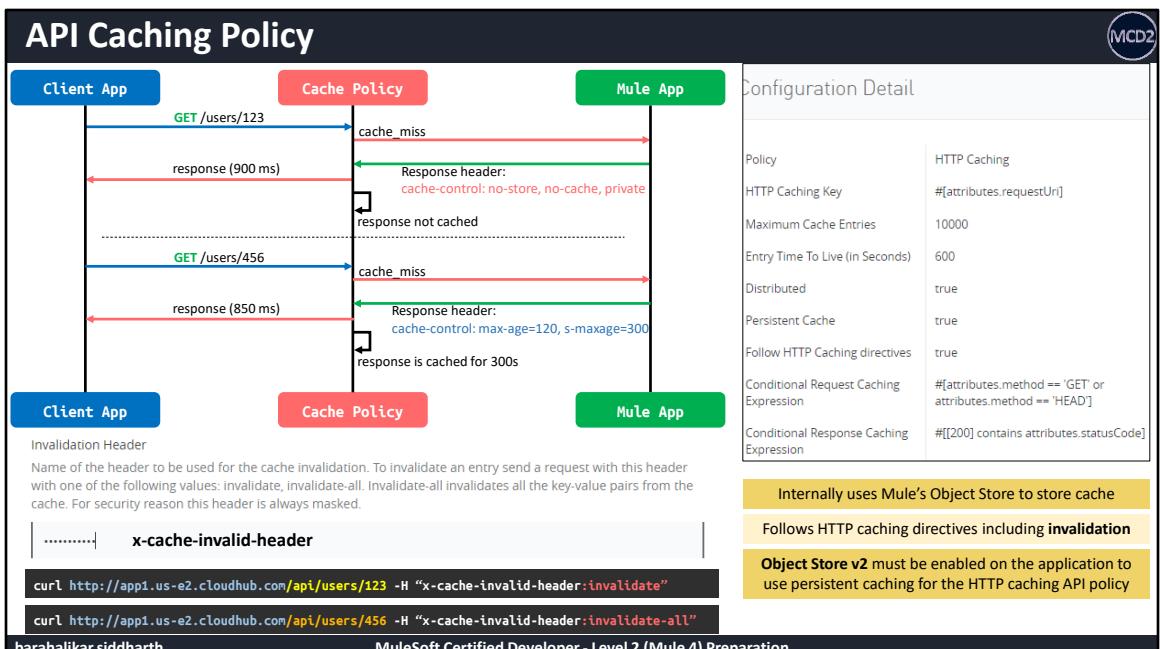
MCD2

Server-Side Caching

- API Caching Policy
- Invalidate Header

YOU SHALL NOT CACHE

barnabasrutherford



So what is api caching policy

Instead of having to write the caching logic inside Mule application code, we can now apply this policy with a single click. With this policy, **users can avoid making multiple calls to a backend system to fetch any data which, in turn, improves API performance by reducing any expensive data processing.**

The API Policy

Internally uses Mule's Object Store to store cache

Follows HTTP caching directives including **invalidation**

Requeries Object Store v2 to be enabled on the application

Within the policy we get options to define the

Cache key

TTL

Persistent

Distributed > so that multiple mule workers share same cache

Conditional request and response expression

The policy also supports an INVALIDATION Header, which can be configured to invalidates the entries in the cache.

We need to define a unique header name, and can be used to either invalidate a single entry or the entire cache

In this curl cmd we use a header value invalidate which removes a single entry
If we use invalidate-all as a header value, the entire cache will be removed

Lets understand what are HTTP CACHING DIRECTIVES, and how it works

The **Cache-Control** HTTP header field holds *directives* which are (instructions) — in both requests and responses — that control caching in shared caches

Lets assume we have a client app, making calls to a Mule application sitting behind a Caching policy

When the client make a GET /users/123 request, the cache server will forward the request to the mule app, because this is the very first call and there is nothing there in the cache.

The Mule app responds back with payload and a response header – cache-control : no-cache, no-store, private

Cache policy will check this response header and will not cache the response

When the client make a 2nd call to GET /users/456 request, the cache server will forward the request to the mule app

Mule app responds back with payload and a cache-control header of max-age=120 and s-maxage=300, both these value are used to define TTL and if both are present, the s-maxage will take precedence

So based on the response header the cache policy will cache the data for 300s

2.13 Sever-Side Caching

- API Caching Policy
- Invalidate Header
- Visulaizer



barahillercardinanth

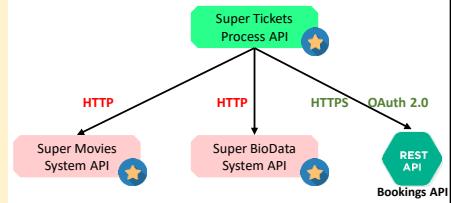
MCD2

Section 2 - Completed

MCD2

#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



barahalikar.siddharth

MuleSoft Certified Developer - Level 2 (Mule 4) Preparation

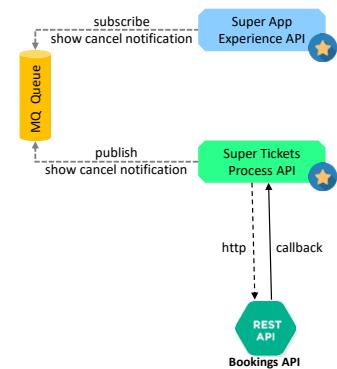
- Welcome to Section2, in this section we will start adding the integration logic to our Super tickets Papi
- We will add system api rest connectors in studio
- Get credentials, apply non functional requirement and invoke SAPI
- We will discuss about Oauth in detail and configure http request connector to call an oauth service to get tokens and use the token in header
- Watch Oauth server interactions using a DEBUG Logger
- To improve performance we will use Scatter gather for parallel execution of these apis, followed by
- Handling errors for scatter gather and the Mule application following the concepts of transient and permanent http errors
- We will end this section by implementation both Client side and server side caching.

Section 3

MCD2

#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



barahalikar.siddharth

MuleSoft Certified Developer - Level 2 (Mule 4) Preparation

Welcome to Section 3, in this section we will discuss about asynchronous message processing

Configure http callbacks and register them with external services

We will configure VM connectors to process messages asynchronously by using DLQs, transactions and redelivery policies.

We will then configure Anypoint mq to publish messages, which will be subscribed by an EAPI

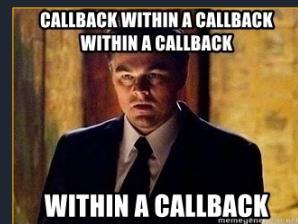
We will improve the performance by configuring a circuit breaker.

We will also looking into CorrelationIDs, Operational logging, Mapped diagnostic context using Tracing and

End the section with validation modules

HTTP Callback

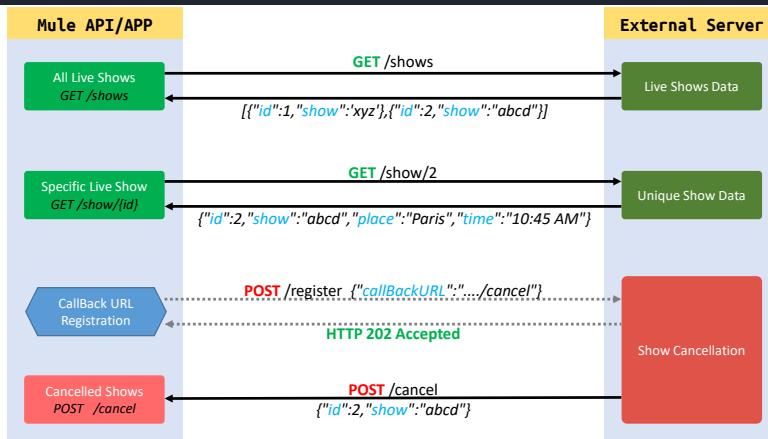
- What/Why?



barahamcaronduh

HTTP Callbacks/Webhooks

MCD2



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

barahalikar.siddharth

MuleSoft Certified Developer - Level 2 (Mule 4) Preparation

What is HTTP Callback?

In Simple terms,

A callback function is like a **Employee** who "calls back" to his **Manager** when he has completed a **Task**.

A **webhook** is a fancy name for an **HTTP callback**.

Lets assume we have a Mule APP and a external server

- The mule app makes calls to the external server to fetch the data about Live Shows
 - The server responds with a JSON Array
- The Mule apps make another call to external server endpoint to fetch unique show details based on a id.

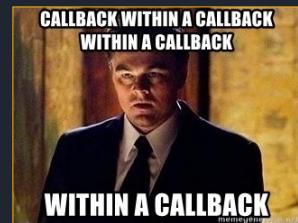
These are regular HTTP GET calls.

- Whenever a show gets cancelled the external server has all the details about it.
- We do not want the mule application to keep on polling the external server to get the show cancellation events, because no one knows when a show will be cancelled.
- Polling can be avoided by using HTTP callbacks.

- **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 responds with a 202 Accepted
 - Whenever a show gets cancelled the **External Server** sends HTTP POST requests to that **Callback URL**
 - **In** this way the mule app receives cancelled show data
-
- And that's how an http callback works

3.1 HTTP Callback

- Configure in Mule Application

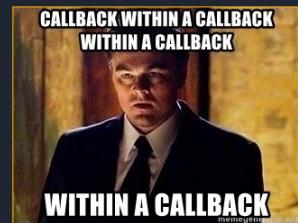


barahamcaronduh

MCD2

3.2 HTTP Callback

- Register Callback with external service
- CloudHub Reserved Properties



barahamercanthanh

MCD2

Asynchronous Processing

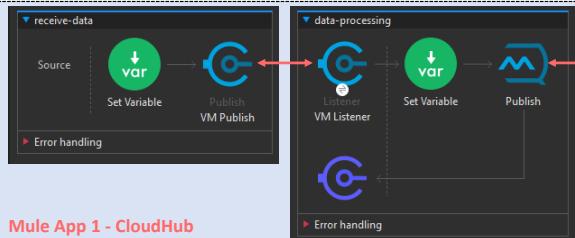
- What/Why?
- VM vs Anypoint MQ



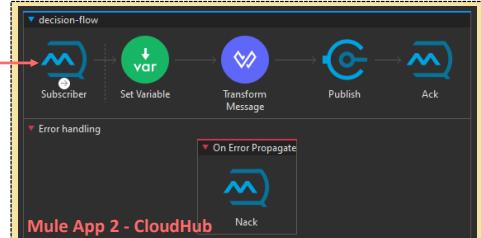
barahamkarandhark
barahamkarandhark

Asynchronous Message Processing

MCD2



- 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



barahalikar.siddharth

MuleSoft Certified Developer - Level 2 (Mule 4) Preparation

- Asynchronous processing is the opposite of synchronous processing, as **the client does not have to wait for a response after a request is made, and can continue other forms of processing.**
- This process is referred to as ***non-blocking*** because the execution thread of the client is not blocked after the request is made.
- In a mule app a Flow Reference passes events **synchronously** between flows
- If data needs to be asynchronously passed between flows within the same application we can make use of VM connector.
- A (VM Connector) manages intra-app and inter-app communication through either transient or persistent 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**
- By using VM queues we can
 - Achieve message reliability

- Distribute work across multiple CloudHub workers and
 - Communicate with another application running in the same Mule domain
- VM module has

• **VM:Publish** is one-way operation – fire & forget

• **VM:Publish consume** is blocking request/response operation

- Okay, now lets say the message has been persisted in a queue and processed. As a next step this processed message needs to be used/consumed by another app. How can this be achieved?
- We can make use of JMS or **Anypoint MQ** connector to pass messages between flows and Mule/Non-Mule applications.
- Anypoint MQ is fully managed cloud messaging system
- To access it all we need is a,
 - URL, client id and secret
- Anypoint MQ includes features:
 - **Queues and Message Exchanges – to send** messages to queues, pull messages from queue
 - This can be done using Simple queuing, pub/sub using exchange and FIFO
- In anypoint mq, The payload format could be JSON,CSV, TEXT
- To handle failing message we can make use of a circuit breaker and
- It provides various options to Acknowledge the events.

VM Queues

MCD2

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

```
1 <vm:config name="vmConfig">
2   <vm:queues>
3     <vm:queue
4       queueName="example-q"
5       queueType="PERSISTENT"
6       maxOutstandingMessages="10" />
7     <vm:queue
8       queueName="example-dlq"
9       queueType="TRANSIENT"
10      maxOutstandingMessages="100" />
11   </vm:queues>
12 </vm:config>
```

```
13 <vm:publish
14   config-ref="vmConfig"
15   queueName="example-q"
16   timeout="5000"
17   timeoutUnit="MILLISECONDS"
18   sendCorrelationId="ALWAYS" />
19
20
21
22
23
24
25
```

```
26 <vm:listener
27   config-ref="vmConfig"
28   queueName="example-q"
29   numberConsumers="2"
30   timeout="5000"
31   timeoutUnit="MILLISECONDS"
32   transactionalAction="ALWAYS_BEGIN">
33   <redelivery-policy
34     idExpression="#{correlationId}"
35     maxRedeliveryCount="4" />
36 </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 redeliver 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

barahalikar.siddharth

MuleSoft Certified Developer - Level 2 (Mule 4) Preparation

Talking about VM queue

A **queue** is a staging area that contains messages that have been sent and are waiting to be read

A **DLQ** – Dead Letter Queue is the same as any other queue except that it receives only undelivered messages

We can configure a queue to be transient or persistent, Reliability pattern strongly suggests that the queues should be persistent

It is always recommended to 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 by DEFAULT within VM connector as well

We can start a TRANSACTION in VM listener, The local transaction starts with the de-

queuing of the message

If any message fails, the 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

**BEST PRACTICES: SYNCHRONOUS
RESPONSE FROM ASYNCHRONOUS PROCESSING**



barahamkarandhark
barahamkarandhark

MCD2

3.4 Asynchronous Processing

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



MCD2

barahamkarandhark
barahamkarandhark

3.5 Asynchronous Processing

- VM Connector
 - Re-delivery Policy

BEST PRACTICES: SYNCHRONOUS
RESPONSE FROM ASYNCHRONOUS PROCESSING



barahamkarandhark
barahamkarandhark

MCD2

3.6 Asynchronous Processing

- VM Connector
 - Add a DLQ
 - Error Handling



barahimkarandanj

MCD2

Anypoint MQ

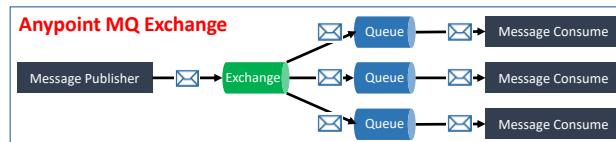
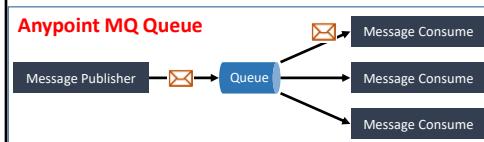
- What/How?
- Queue vs Exchange
- Steps



barahmierandmorth

Anypoint MQ

MCD2



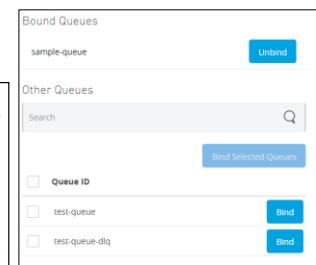
Anypoint MQ Queue

- 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



barahalikar.siddharth

MuleSoft Certified Developer - Level 2 (Mule 4) Preparation

https://www.youtube.com/watch?v=Uz5S20NHpls&ab_channel=MuleSoft-TechZone

As discussed earlier an **Anypoint MQ** connector can pass messages between flows and Mule/Non-Mule applications.

In Anypoint MQ Queue

A Publishers can publish messages into queues, and any one consumer can receive messages from queue.

There are multiple options,

A Standard queue can

- Send & receive messages unordered (does not guarantee order)
- It is a Best fit for fast delivery, but only when order is not important
- In FIFO (First in First Out)**
 - Ordering of messaging is maintained

- It has a Built-in deduplication of messages (Exactly Once Delivery)
- **Then we have the DLQ (Dead Letter Queue)**
 - It has the Same queue type, region, encryption as origin queue
 - Anypoint MQ does 10 reroute attempts before sending payload to DLQ
 - Recovery of message in DLQ is done via MQ Administration API
 - A dlq Can be associated with one or more queues
- Message **Format & Size**
 - Text, JSON or CSV format message
 - Maximum 10MB message size

With **message exchanges** you can distribute a single message to multiple consumers.

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

These images show how we can create queue, exchange, FIFO queues

And how to Bound queues to a message exhcnage

Anypoint MQ - Steps

The screenshot shows the Anypoint MQ interface with three main panels:

- Create Queue:** A form to create a new queue named "demo-queue". It includes fields for Message TTL (7 Days), Default Acknowledgement Timeout (2 Minutes), and various delivery and security options (Delivery Delay, Encryption, Dead Letter Queue). A "Create Queue" button is at the bottom.
- Message Sender:** A panel for sending messages. It shows a message being sent with Type "Text", Payload "testing", and Delivery Delay set to 0 seconds. A "Message Sender" button is present.
- DemoClientApp:** A panel for managing a client application. It shows a Client App ID ("391f7b6be93b497e83a5a69a31ae95e0") and a Client Secret. A note states that clients have attributes like name, client id, and client secret, and cannot access queues across environments. It also shows XML configuration code for the client.

Anypoint MQ standard and FIFO queues are available in 10 regions at the time of this recording.

We can select any one of them.

Create a queue by giving it a id, configure Message TTL and ACK timeout, default is 2mins

We can also configure delivery delay, encrypt the data and assign a DLQ

Within the UI we have the option to send and browser messages. We can also make use of Anypoint MQ REST APIs for all these operations.

To access MQ, we need to create a Client app, and use the credentials.

Clients cannot access queues across different environments

Within each environment a new client should be created

In Anypoint Studio we can configure URL, Client id and secret to access the queues.

3.7 Anypoint MQ

- Configure Anypoint MQ in Mule app



barahimkarandam

(MCD2)

3.8 Anypoint MQ

- Publish Messages to Anypoint MQ



barahimkarandamir

MCD2

Anypoint MQ

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



barahamercanthan

Message Acknowledgement

MCD2

Acknowledgement Types	Acknowledgement Modes
<ul style="list-style-type: none"> ACK <ul style="list-style-type: none"> Indicates that an application has processed the message Messages will be deleted from Queue NACK <ul style="list-style-type: none"> Indicates that an application did not process the message Messages remain in Queue acknowledgementTimeout (optional) <ul style="list-style-type: none"> time that Anypoint MQ waits to receive an ACK or NACK default TTL is 2 minutes Applicable for, <ul style="list-style-type: none"> MANUAL or AUTO acknowledgment modes 	<ul style="list-style-type: none"> AUTO (default) <ul style="list-style-type: none"> 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 <ul style="list-style-type: none"> Responsibility of application logic to manage ACK/NACK requires a AnypointMQMessageContext object IMMEDIATE <ul style="list-style-type: none"> 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**

Anypoint MQ lets you determine how to process messages in Anypoint Studio using the Anypoint MQ connector.

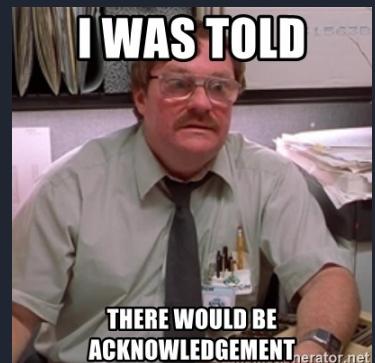
Two acknowledgment responses can occur:

- Acknowledgment (ACK)**
 - Indicates that an application has processed the message
 - Messages will be **deleted** from Queue
- Negative acknowledgment (NACK)**
 - Indicates that an application did not process the message
 - Messages **remain** in Queue
- There is another option, **acknowledgementTimeout (optional)**
 - It is the time that Anypoint MQ waits to receive an ACK or NACK
 - default TTL is 2 minutes
 - Applicable for,
 - MANUAL or AUTO acknowledgment modes

- It also has **Acknowledge modes**
- **AUTO** (default)
 - Sends ACK only if flow execution completes successfully
 - Sends NACK if error occurs in flow, message returned to queue
 - Only applicable for Subscriber message source
- **MANUAL**
 - Responsibility of application logic to manage ACK/NACK
 - requires a **AnypointMQMessageContext** object
- **IMMEDIATE**
 - Direct ACK when message is consumed, prior of any processing
 - Message gets removed from queue
 - All these can be configured using anypoint mq connectors in studio
- We also have an **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

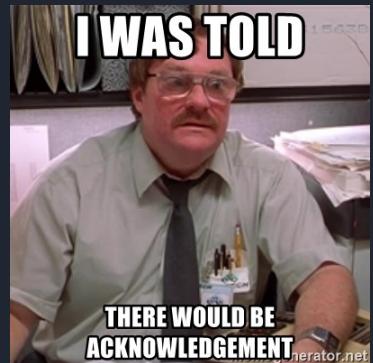


MCD2

barahamercanthith

3.10 Anypoint MQ

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

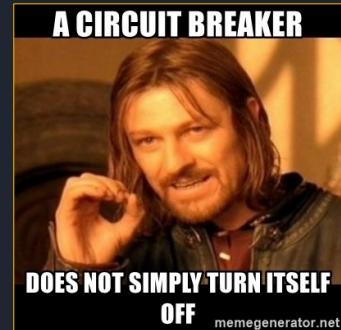


MCD2

brianmcarthur

Anypoint MQ

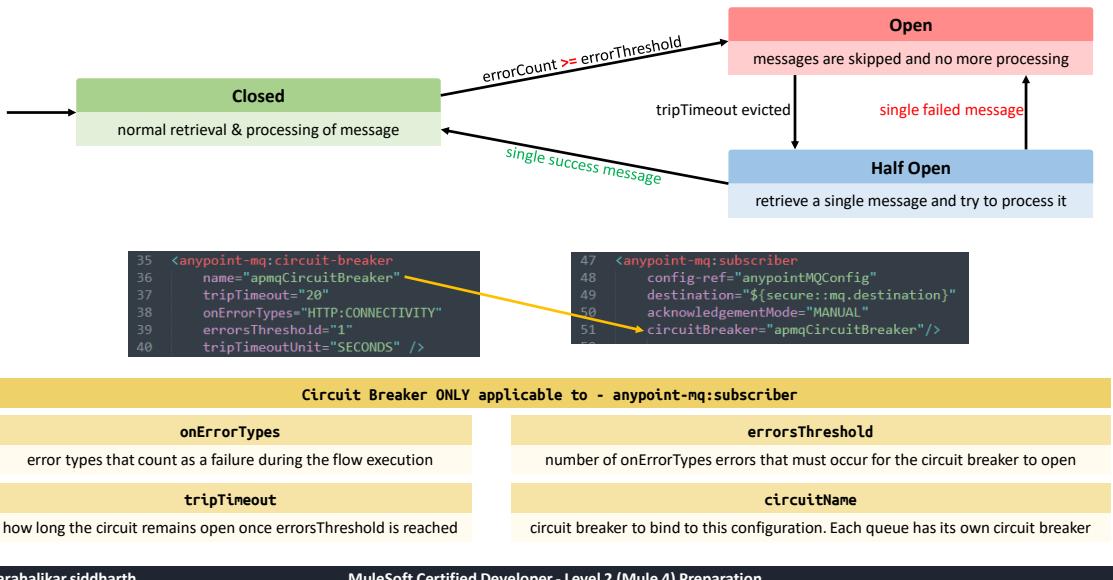
- Circuit Breaker
 - Configuration



barahamercanthan

Anypoint MQ – Circuit Breaker

MCD2



A circuit breaker is an electrical switch designed **to protect an electrical circuit from damage caused by overcurrent/overload or short circuit.**

In software, a Circuit breaker is used to detect failures and to prevent a failure from constantly recurring, during temporary external system failure.

Anypoint Mq circuit breaker is only applicable to MQ Subscriber operation

In circuit breaker terminology we have 3 states closed, open and half-open

- **Closed** state is When everything is normal, the circuit breakers remained closed, and all the request passes through to the services
- in **Open state** circuit breaker returns an error immediately without even invoking the services. .
- in **Half Open state**, the circuit breaker allows a 1 requests from the client to passthrough and invoke the external service. If the requests are successful, then the circuit breaker will go to the closed state. However, if the requests continue to fail, then it goes back to Open state.

In Studio we configure a circuit breaker to respond to certain error types like HTTP

CONnectivity or NOT FOUND and add it to the MQ Subscriber

When a error occurs the CB checks if the `errorCount >= errorThreshold`. If it is true then it goes into Open state

OnErrorTypes defines error types that count as a failure during the flow execution

errorsThreshold - is number of **onErrorTypes** errors that must occur for the circuit breaker to open

tripTimeout configuration is how long the circuit remains open once errorsThreshold is reached

Once the trip timeout is evicted, the CB goes into Half open state

When it is in half open state the CB sends a single event to the external service and if fails, the CB goes into a OPEN state,

If it is success it goes into a Closed state.

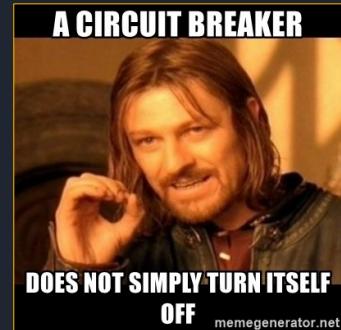
After configuring the circuit breaker, we need to configure MQ Subscriber with the circuitbreaker config

It is always recommended for each queue to have its own circuit breaker.

In the next hands on session we will implement this.

3.11 Anypoint MQ

- Circuit Breaker



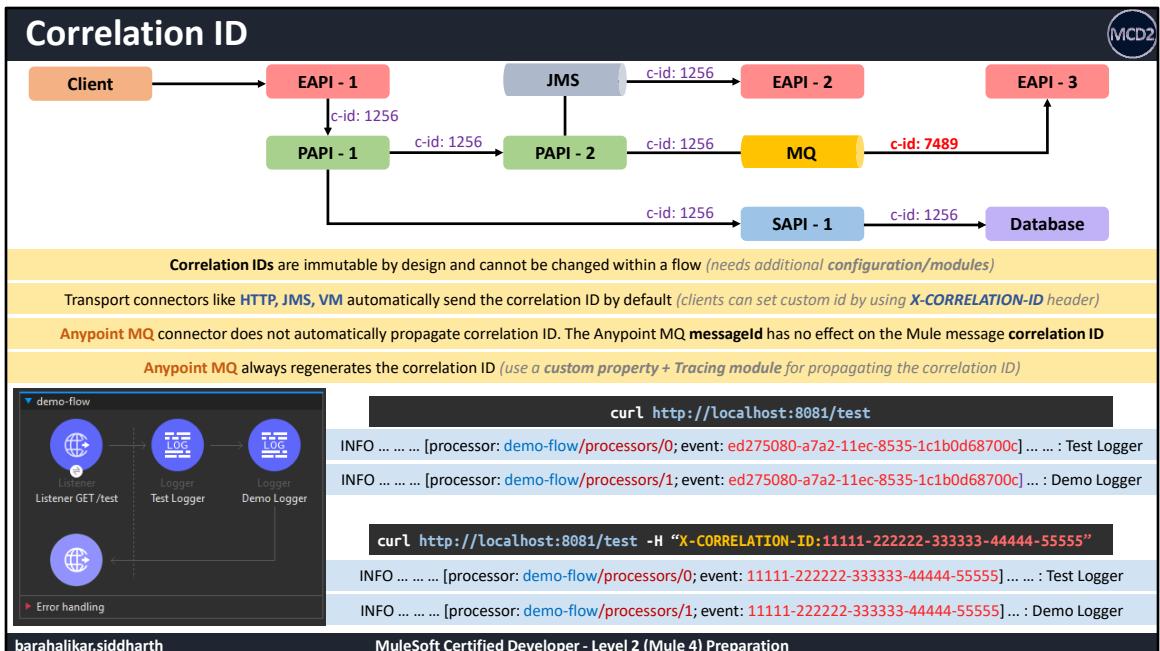
MCD2

Correlation ID

- What?
- How?



barahamercardonth



In a distributed system architecture, it is highly difficult to understand a single end to end customer transaction flow through the various components.

To solve this we use a [Correlation Pattern](#), It is a well documented Enterprise Integration Pattern.

This pattern depends on the use of Correlation ID.

A Correlation ID is a unique identifier that is added to the very first interaction (incoming request) to identify the context and is passed to all components that are involved in the transaction flow. Correlation ID becomes the glue that binds the transaction together and helps to draw an overall picture of events.

- *In this image we have a set of EAPI, PAPI and SAPI, communication through HTTP, JMS and MQ connectors.*
- *Since the client is not sending any correlation id,*
- *Mule application's HTTP Listener automatically generates a correlation ID for the Mule event and propagates it across the calls.*

- **Correlation IDs** is a simple **UUID** but immutable by design and cannot be changed within a flow (*it needs additional configuration/modules to do so*)
- Transport connectors like **HTTP, JMS, VM** automatically send the correlation ID by default
- When a correlation ID has been set, Mule will preserve it across transport. This means that Mule will propagate the correlation ID alongside the message payload.
- *But what about Anypoint MQ connector*
- **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 (*we can use a **custom property + Tracing module** for propagating the correlation ID, we will see more on this in another session*)
- *So if you notice the image, only MQ connector uses a new corealition id compared to others*
- *In the bottom image we have a simple Mule flow listening on HTTP GET /test endpoint with 2 logger,*
- *When we hit it without any correlation header, Mule http connector will automatically generate and use a correlation id.*
 - *This can be seen on the log message event details*
 - *It used a same correaltionId for both loggers*
- *If required the client can set a custom correlation id in the header and mule will preserve and propogate that id across the flows*
 - *The header name could be - X-CORRELATION-ID*

3.12 Correlation ID

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



barahmierandforth

MCD2

Logging & Tracing

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



barahamkaranthith

Logging & Tracing Module

MCD2

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*)

Synchronous Logging: thread waits during I/O operation → LOG

Asynchronous Logging: thread DOESNT wait during I/O operation → LOG

<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

barahalikar.siddharth MuleSoft Certified Developer - Level 2 (Mule 4) Preparation

- Mule runtimes and Mule applications use standard log4j2 logging
- Logs help to debug and track processing of Mule applications, such as details from Mule events
- Logging component creates log messages with various log levels, such as INFO, WARN, ERROR, and DEBUG
 - By default, Mule runtimes show the INFO log level, and ignore DEBUG or TRACE level log messages
- Log messages can be configured to log to various appenders – like system console, a file, a database, the CloudHub logging service
 - CloudHub **ignores** and **overrides** the application's **log4j2** configuration (*optionally we can Disable CloudHub logs and log to external systems as well*)
- It Supports both synchronous or asynchronous logging
 - By default, logging in Mule is done asynchronously
- In Synchorunus logging
 - The execution of the thread processing event is interrupted to wait for the

log message

- It degrades application Performance and
 - It is used in logging CRITICAL messages
- In asynchronous logging
 - The logging operation occurs in a separate thread
 - So improve apps performance, but
 - Log may be lost in case of application crash
- The standard log4j2 patternlayout logs the processerPath and the correlation id in the log context. This is the out of the box pattern layout
- We can use Mapped Diagnostic Context (MDC) to enrich logging and improve tracking by providing more context or information in the logs for the current Mule event.
- But how do we use MDC?
- It is done using Tracing Module?
 - What is Tracing module?
 - Tracing module enables you to enhance your logs by adding, removing, and clearing variables from the logging context for a given Mule event.
 - It also enables you to modify the correlation ID during flow execution.
 - Lets take a example, in this image we have a flow with 3 loggers, Test, Demo, Tracing Logger
 - We are also making use of tracing operations to provide more information in the logs,
 - We have added 2 set logging variable operations to set some info in the log context
 - Like the querparamter valye and the requestPath
 - Then we use a remove logging context to remove one of the set logging variable
 - Finally we use a Clear logging variable to removes all variables set by the tracing module
 - When we hit the app with a GET method on /test endpoint with a ID queryparameter, this is how logs looks like
 - The 1st logger logs TEST Logger message, but see the log context it also includes id and requestPath along with correaltionId and processpath
 - Similarly in the 2nd log message we see DEMO LOGGER message and in the log

context we only see id=007, because we have removed the requestPath

- In the final log, we don't see any special variable because we cleared all logging variables
- This is how a tracing module can be used to enrich the logging context

3.13 Logging & Tracing

- Tracing Correlation ID across Anypoint MQ



barahamercanthanh

(MCD2)

3.14 Logging & Tracing

- Mapped Diagnostic Context

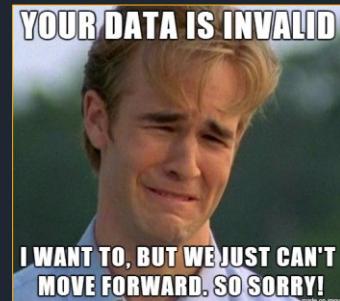


barahamercanthan

(MCD2)

Validation Module

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



barahamercardonth

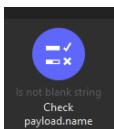
Validation Module

MCD2

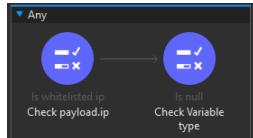
For APIs, **APIKit router** validates requests with API Specifications (DataTypes/Schemas) in RAML/OAS/WSDL

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

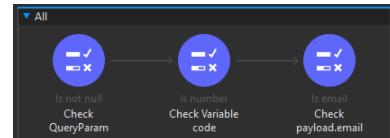


Is not blank string
Check payload.name



Is whitelisted ip
Check payload.ip

Is null
Check Variable type



Is not null
Check QueryParam

Is number
Check Variable code

Is email
Check payload.email

```
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>
```

Operations
Is IP
Is URL
Is blank string
Is elapsed
Is email
Is empty collection
Is false
Is not blacklisted ip
Is not blank string
Is not elapsed
Is not empty collection
Is not null
Is null
Is number
Is time
Is true
Is whitelisted ip
Matches regex
Validate size
Scopes And Routers
All
Any

barahalikar.siddharth

MuleSoft Certified Developer - Level 2 (Mule 4) Preparation

Validation module verifies that the content of a message in a Mule flow matches specific criteria. If a message does not meet the defined validation criteria, the validation fails and returns a validation error.

Why use a validation module?

For APIs we have a **APIKit router** which validates requests with API Specifications (DataTypes/Schemas) in RAML/OAS/WSDL

Any endpoint which is not part of the API specification like JMS,VM,MQ endpoint, we use the Validation Module to validate messages.

It can be Used for pre/post conditions validation and raises standard validations error or custom errors as well

In the future we will be using the Async APIs.

The validation module has a rich set of operations.

Which can be used individually or as part of a scope

If used within a All scope, all opeations must pass to proceed futher

If used within a Any scope, than at least one operation should pass to proceed futher

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

Message : Input XML was not compliant with the schema.

Error type : XML-MODULE:SCHEMA_NOT_HONOURED

```
1 <json:validate-schema  schema="schemas/JsonDemoSchema.json" />
2 {
3   "$schema": "http://json-schema.org/draft-07/schema",
4   "$id": "http://example.com/example.json",
5   "type": "object",
6   "examples": [ ],
7   "required": [
8     "title", "tool"
9   ],
10  "properties": {
11    "title": { },
12    "tool": { }
13  },
14  "additionalProperties": true
15 }
```

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

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

Message : Json content is not compliant with schema.

Error type : JSON:SCHEMA_NOT_HONOURED

barahalikar.siddharth

MuleSoft Certified Developer - Level 2 (Mule 4) Preparation

In mulesoft, we can use XML module to **Validate schema**, it validates that the input content is compliant with a given XSD schema.

In this example we have xsd which accepts 2 xml elements title and tool of type string. When the user sends a payload, the xml module validates it againsts the schema and allow it because it is complaint.

When the user sends a payload which has a extra element that is not defined in the schema, XML module raises an error.

- This is where we talk about an **integration pattern called as** a Tolerant Reader, which **helps creating robust communication systems**.
- The idea is to be as tolerant as possible when reading data from another service.
- We only validate required fields to make integrations more resilient to changes from other systems

It has been explained in depth by Martin Flower in one of his blog, I have added a link here, please go ahead and read it.

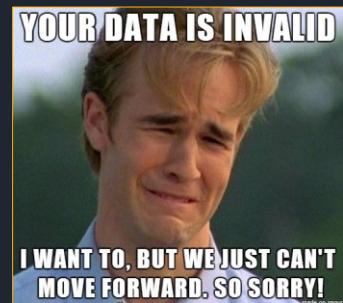
To achieve this, within XML XSD, we can add a lax element, which will not raise an error if schema cannot be obtained for a particular element

=====

- On similar grounds we can also validate JSON payloads using a JSON schema,
- In JSON schema, we make use of the additionalProperties field to allow extra properties in a JSON Payload
- If additionalProperties is set to false, the schema won't allow any additional properties and will raise an Error

3.15 Validation Module

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

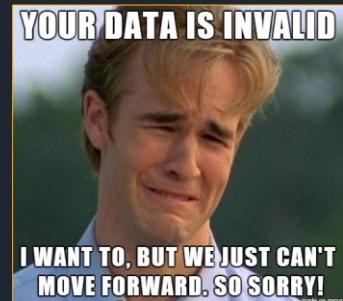


barahamkaranthanth

MCD2

3.16 JSON Schema Validation

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

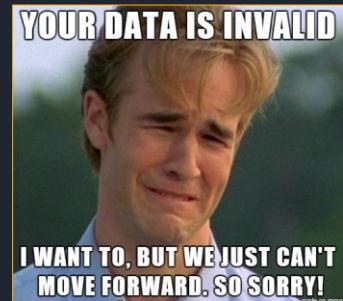


barahamercanthan

MCD2

3.17 XML Schema Validation

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



barahamercanthanh

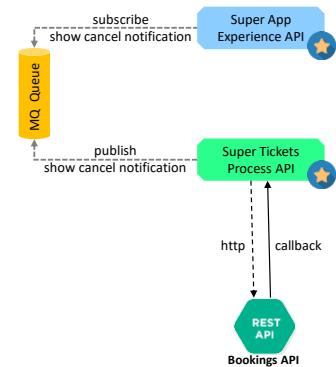
MCD2

Section 3 - Completed

MCD2

#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



barahalikar.siddharth

MuleSoft Certified Developer - Level 2 (Mule 4) Preparation

Welcome to Section 3, in this section we will discuss about asynchronous message processing

Configure http callbacks and register them with external services

We will configure VM connectors to process messages asynchronously by using DLQs, transactions and redelivery policies.

We will then configure Anypoint mq to publish messages, which will be subscribed by an EAPI

We will improve the performance by configuring a circuit breaker.

We will also looking into CorrelationIDs, Operational logging, Mapped diagnostic context using Tracing and

End the section with validation modules

Section 4

MCD2

#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

barahalikar.siddharth

MuleSoft Certified Developer - Level 2 (Mule 4) Preparation

- Welcome to Section 4
- This is a small but exciting section where
- we will configure health endpoint for our Mule application
- Understand liveness and readiness concepts
- Create a new mule library with common application logic
- Install it to local maven repo
- We will also create a new mule connector using XML sdk,
- Build and publish it to exchange
- And end the section by monitoring the APIs using health endpoints

Health Endpoints

- Liveness Probe
- Readiness Probe
- API Monitoring



barahamkaranthmuth

Health Endpoints

MCD2

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


```

graph TD
    subgraph API_Monitoring_Tool [API Monitoring Tool]
        direction LR
        subgraph API_Functional_Monitoring [API Functional Monitoring]
            direction TB
            PAPI1[PAPI - 1] --> Live1[/live]
            PAPI1 --> Ready1[/ready]
            PAPI2[PAPI - 2] --> Live2[/live]
            PAPI2 --> Ready2[/ready]
            PAPI3[PAPI - 3] --> Live3[/live]
            PAPI3 --> Ready3[/ready]
        end
        subgraph External_Tools [External Tools]
            direction TB
            SAPI1[SAPI - 1] --> LiveS1[/live]
            SAPI2[SAPI - 2] --> LiveS2[/live]
            SAPI3[SAPI - 3] --> LiveS3[/live]
        end
        subgraph Scripts [Scripts]
            direction TB
            SAPI1 --> LiveS1
            SAPI2 --> LiveS2
            SAPI3 --> LiveS3
        end
    end
    PAPI1 --> SAPI1
    PAPI2 --> SAPI2
    PAPI3 --> SAPI3
    SAPI1 --> SAPI2
    SAPI2 --> SAPI3

```

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

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

Health Endpoint is a common practice in building APIs. It is a **special REST API implementation** that you can use to validate the status of a application and its dependencies. It responds back with **the HTTP status that defines whether the service is "Up or Down"**

We use a liveness probe with a /live endpoint

To check that application is deployed and responding to simple HTTP calls (*if fails, restarting the application might help*)

We use a readiness probe with /ready endpoint

To check **liveness** of downstream dependencies to route real traffic (*if fails, restarting the application will not help solve the issue*)

These endpoints are monitoring by an API Monitoring tool

API monitoring is the process of **observing the functional, availability, and performance metrics of an API**. It can Optionally sends out reports about whether tests pass or fail.

We can setup a simple api monitor using Anypoint Functional monitoring or any other external tool/scripts by

- defining the endpoint to monitor,
 - adding option request details,
 - Verifying assertions like statuscode, responses
 - Optionally sending reports to email, slack channels
-
- Lets assume we have a process api which depends on SAPI.
 - All apis should have these health endpoint defined
 - An api monitor first checks the liveness of PAPI,
 - If it fails, PAPI will not able to server traffic even if the SAPI is live
 - restarting the PAPI might help in this case
 - In the 2nd example,
 - The PAPI liveness check has passed but
 - The downstream SAPI liveness check failed
 - In this case restarting the PAPI will not help and
 - PAPI cannot server traffic
 - In the 3d example,
 - All the health checks pass for both PAPI and SAPI
 - PAPI will be able to server traffic

4.1 Health Endpoints

- Liveness Probe
- Readiness Probe
- HTTP Status Code Validator

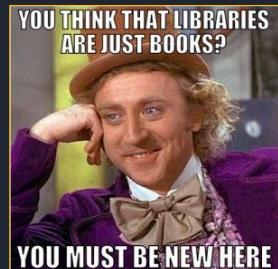


barahimkarandamir

MCD2

Shared Libraries

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



YOU THINK THAT LIBRARIES
ARE JUST BOOKS?

YOU MUST BE NEW HERE

barnabasandforth

Share Mule Artefacts as Library

MCD2

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

```
1 ...
2 <groupId>com.super.org</groupId>
3 <artifactId>common-app-logic</artifactId>
4 <version>1.0.0</version>
5 <packaging>mule-application</packaging>
6 ...
7 <plugin>
8   <groupId>org.mule.tools.maven</groupId>
9   <artifactId>mule-maven-plugin</artifactId>
10  <version>3.3.9</version>
11  <configuration>
12    <classifier>mule-plugin</classifier>
13  </configuration>
14 </plugin>
15 ...
16 <dependency>
17   <groupId>org.mule.connectors</groupId>
18   <artifactId>mule-http-connector</artifactId>
19   <version>1.6.0</version>
20   <classifier>mule-plugin</classifier>
21 <scope>provided</scope>
22 </dependency>
23 ...
24
25 <!-- src/main/mule (Flows)
26   -->
27   <!-- error-common.xml
28   -->
29   <!-- health-common.xml
30   -->
31   <!-- stubs.xml
32 -->
```

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

Mule EAPI/PAPI/SAPI

```
26 <dependency>
27   <groupId>org.mule.connectors</groupId>
28   <artifactId>mule-http-connector</artifactId>
29   <version>1.6.0</version>
30   <classifier>mule-plugin</classifier>
31 </dependency>
32
33 <dependency>
34   <groupId>com.super.org</groupId>
35   <artifactId>common-app-logic</artifactId>
36   <version>1.0.0</version>
37   <classifier>mule-plugin</classifier>
38 </dependency>
```

mvn clean verify -U

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

barahalikar.siddharth

MuleSoft Certified Developer - Level 2 (Mule 4) Preparation

Reusability is extremely important to deliver functionality quicker and in a more consistent manner.

Within Mulesoft we can create a shared library which contains flows, sub-flows,, which can be re-used across the Mule projects. For example, common sub-flows can be used for logging, error-handling, shared business logic, etc.

To create a shared library, We start of by creating a normal mule app, in pom xml we must make sure that the mule maven plugin classifier is of type **mule-plugin**

If the application/library depends upon any connector, we can define the scope as provided

This means that the dependency should be provided by the application using this library

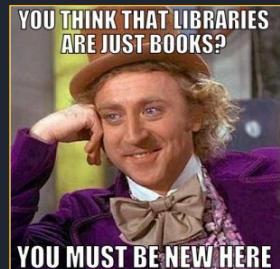
Then we can add the common business logic and build the application
It can be installed to local maven repo or deployed to exchange as well

Any application can now use this library by defining it in the dependencies section

And use the common functionality using the IMPORT feature.

4.2 Shared Libraries

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



YOU THINK THAT LIBRARIES
ARE JUST BOOKS?

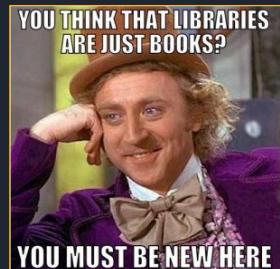
YOU MUST BE NEW HERE

barahamercandorith

MCD2

4.3 Using Shared Library

- Configure it in a Mule app
- Test



barahmierandforth

MCD2

Custom Connector Development

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



barahamkarandharkh

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
```

liveness-check-extension
 src
 main
 resources
 org
 mule
 extensions
 smart
 connector
 module-liveness-check.xml
 test
 munit
 assertion-munit-test.xml
pom.xml

```
1 <module>
2   <operation>
3     <parameters>
4       <parameter></parameter>
5     </parameters>
6   <body></body>
7   </operation>
8 </module>
```

```
1 <module>
2   <operation>
3     <parameters>
4       <parameter>
5         name="url"
6         displayName="URL"
7         type="string"
8         use="REQUIRED" />
9     </parameters>
10    <body>
11      <http:request
12        config-ref="livenessHttpRequestConfig"
13        method="GET"
14        url="#{vars.url}">
15        <mule:error-mapping
16          sourceType="HTTP:UNAUTHORIZED"
17          targetType="LIVENESS-CHECK:UNAUTHORIZED" />
18      </http:request>
19      <mule:set-payload value="#{true}" />
20    </body>
21  </operation>
22 </module>
```

mvn clean install (installs to local m2 repository)

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

barahalikar.siddharth

MuleSoft Certified Developer - Level 2 (Mule 4) Preparation

MuleSoft XML SDK lets developers write custom connectors and is actually easier than using the Java-based Mule SDK.

XML SDK Creates a mule-plugin artefact and the Exchange Mule Maven Plugin is used for deployment to Exchange (*mvn deploy*)

We will be developing a custom connector for the Dependency/Downstream API Liveness Check.

To create an XML SDK module:

We start the process by Add the MuleSoft repository to your Maven (mvn) settings file

And use mvn archetype command to generate the Mule Project Skeleton

Lets see what attributes are defined in this command

archetypeGroupId

The GroupId in the Mule repository where the skeleton of the XML SDK is defined.

archetypeArtifactId

The ArtifactId of the archetype in the Mule repository.

archetypeVersion

A version of the archetype.

groupId

The GroupId for your application or module. Generally, it should be an AnypointOrganizationId.

artifactId

The ArtifactId for your project or module.

muleConnectorName

The name of your Mule custom connector.

The module-liveness-check.xml file Defines the XML SDK root element and all connector logic is defined here

An XML SDK module has an operations elements.

An <operation> element defines parameter, body, output and error mapping options
Within the operation element

- a set of **input** parameters defines data which is the only data the message processors in the <body> scope can access.
 - A **body** defines a chain of components to be executed, similar to a mule flow.
 - Operation elemenet also has a single **output** which Declares the output type of your XML SDK module and
 - if any error from body should not be propagated as it is, we can do **Error** mapping as well
-
- Can we use mvn cmd to install this to local maven repo or deployinf to exchange

Talking about Limitations

- XML SDK only provides outbound operations, it cannot have sources (such as a <scheduler>) or routers.
- There is No graphical editor and all xml namespaces must be prefixed
- To prevent errors it is recommended to Use the kebab case naming convention to name parameters and operations in any XML SDK project

There is a lot more XML SDK, Please chec the mule documentation for more info and examples

XML SDK – Catalog

MCD2

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

```
liveness-check-extension
  src
    main
      resources
        module-liveness.xml
        module-liveness-catalog.xml
        user-schema.json
      test
    target
    pom.xml
```

```
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>
```

barahalikar.siddharth

MuleSoft Certified Developer - Level 2 (Mule 4) Preparation

What is a Catalog within XML SDK?

- The standard data types for <property> and <parameter> are primitive types: string, boolean, number, date, datetime, localdatetime, time, localtime, timezone, binary, any, regex.
- For Data Types with **complex** structure we can create a **CATALOG** of data type and inject them into the **MODULE**

This example creates a catalog file for JSON schema

The catalog file can reference XSD and JSON schema files

Once the catalog and schemas are ready, you use **types** in the output of a operations
The type is nothing but the name of the catalog

4.4 Custom Connector Development

- XML SDK
- Generating the skeleton
- Make few changes



barahamercanthanh

MCD2

4.5 Custom Connector Development

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



barahamercanthan

MCD2

4.6 Custom Connector Development

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



barahamercanthan

MCD2

4.7 Custom Connector Development

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



barahamkaranthanth

MCD2

Monitoring

- Anypoint Functional Monitoring Steps



barahamkaranthanth

Anypoint Functional Monitoring

MCD2

The screenshot shows the Anypoint Functional Monitoring interface. It consists of several panels:

- Step 1 - Setup the monitor:** Fields include Monitor name (Test-Monitor), Select location (us-east-1), and Monitor schedule (Every 15 minutes).
- Step 2 - Select endpoints:** Endpoint #1 configuration: Method (GET), Endpoint URL (https://httpbin.org/ip), Headers (optional), Assertions (Status code Must equal).
- Step 3 - Set notifications:** Notifier selection: Select..., Slack, PagerDuty, NewRelic, SumoLogic, Email.

Below these panels are three data sections:

- Average response times in milliseconds:** A chart showing response times for Sun, Mon, and Tue. The x-axis ranges from 10 to 40 ms.
- Last executions:** A table showing recent successful executions:

Execution	Time Ago
Passed	5 minutes ago
Passed	6 minutes ago
Passed	20 minutes ago
- Scheduled runs:** A table showing scheduled runs with their next run time, execution location, and status.

Next run	Schedule	Execution Location	Location Status
in 8 minutes	Every 15 minu	us-east-1	ONLINE
in 9 minutes	Every 15 minu	us-east-2	ONLINE

At the bottom are buttons: Create Monitor, Customize Monitor, Download Monitor, and Upload Monitor.

As discussed earlier an API monitoring is the process of **observing the functional, availability, and performance metrics of an API**. Optionally sends out reports about whether tests pass or fail.

For using Anypoint functional monitor we follow these steps

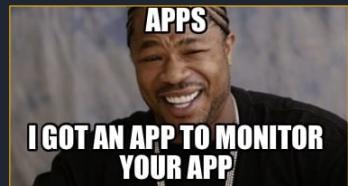
- We create a monitor by giving it a name, select a location to run the tests from and configure a schedule
- In 2nd step we add the endpoints that needs to be monitor and add assertions to check status code and others
- In 3rd step we set the notifications to notify a user when a monitor fails.

We can review the details of past test runs. By accessing the **View Monitor History option**

We have the options to
Customize monitors
Download and upload monitors

4.8 Monitoring

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



barahamkaranthith

MCD2

Section 4 - Completed

MCD2

#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

barahalikar.siddharth

MuleSoft Certified Developer - Level 2 (Mule 4) Preparation

- Welcome to Section 4
- This is a small but exciting section where
- we will configure health endpoint for our Mule application
- Understand liveness and readiness concepts
- Create a new mule library with common application logic
- Install it to local maven repo
- We will also create a new mule connector using XML sdk,
- Build and publish it to exchange
- And end the section by monitoring the APIs using health endpoints

Section 5

MCD2

#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

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['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

Log Country Name

Log Multiple Details

barahalikar.siddharth

MuleSoft Certified Developer - Level 2 (Mule 4) Preparation

Welcome to Section 5

In this section

We develop a custom api poloicy from scratch and apply it using API Manager and also within Anypoint Studio

Custom API Policy

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



barahamercandanth

Custom API Policy - Development

MCD2

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
```

```
1 ...
2 <http-policy:proxy name="{{policyId}}-policy">
3   <http-policy:source>
4     <set-variable
5       variableName="msg"
6       value="{{(message)}}"/>
7     <logger
8       message="#{vars.msg}"/>
9     <http-policy:execute-next/>
10    </http-policy:source>
11  </http-policy:proxy>
12 </http-policy:source>
13 </http-policy:proxy>
14 </http-policy:source>
15 </http-policy:proxy>
```

pom.xml

```
test-policy
└── src
    ├── main
    │   └── mule
    │       └── template.xml
    ├── mule-artifact.json
    └── pom.xml
    └── test-policy.yaml
```

```
20 id: custom-log-message
21 name: Custom Log Message
22 description: Policy for logging
23 category: Custom
24 type: custom
25 resourceLevelSupported: true
26 encryptionSupported: false
27 standalone: true
28 requiredCharacteristics: []
29 providedCharacteristics: []
30 configuration:
31   - propertyName: message
32     name: Enter Message
33     description: |
34       Enter value to Log
35     type: string
36     optional: true
37     sensitive: false
```

```
6 ...
7   <groupId>1a1a1a-2b2b2b-3c3c3c</groupId>
8   <artifactId>test-policy</artifactId>
9   <version>1.0.0-SNAPSHOT</version>
10  <name>test-policy</name>
11  <packaging>mule-policy</packaging>
12 ...
13 <build>
14   <plugins>
15     <plugin>
16       <groupId>org.mule.tools.maven</groupId>
17       <artifactId>mule-maven-plugin</artifactId>
18       <version>${mule.maven.plugin.version}</version>
19       <extensions>true</extensions>
20     </plugin>
21     <plugin>
22       <groupId>org.apache.maven.plugins</groupId>
23       <artifactId>maven-deploy-plugin</artifactId>
24       <executions>
25         <execution>
26           <executions>
27             </executions>
28           </execution>
29         </executions>
30       </plugin>
31     </build>
32 
33 <distributionManagement>
34   <repository>
35     <id>exchange-server</id>
36     <name>Corporate Repository</name>
37     <url>${exchange.url}</url>
38     <layout>default</layout>
39   </repository>
40 </distributionManagement>
41 ...
42 <repository>
43   <id>exchange-server</id>
44   <name>Corporate Repository</name>
45   <url>${exchange.url}</url>
46   <layout>default</layout>
47 </repository>
48 ...
49 <repository>
50   <id>exchange-server</id>
51   <name>Corporate Repository</name>
52   <url>${exchange.url}</url>
53   <layout>default</layout>
54 </repository>
55 ...
56 </distributionManagement>
57 ...
58 <repository>
59   <id>exchange-server</id>
60   <name>Corporate Repository</name>
61   <url>${exchange.url}</url>
62   <layout>default</layout>
63 </repository>
64 ...
65 </distributionManagement>
66 ...
67 ...
```

For applying policy using API Manager

mvn deploy (deploys to Anypoint Exchange)

barahalikar.siddharth

MuleSoft Certified Developer - Level 2 (Mule 4) Preparation

Custom Policies are policies which can be developed and applied to their APIs, to extend the functionality of Mule application or existing API Policies.

Policies can be applied/deployed locally as offline policies.

For applying online, policies should be deployed to Anypoint Exchange

- First step to develop a custom policy consists in setting up a project with the required files.
- The easiest way to gather all your required files is by using the maven archetype.
- This archetype creates all the necessary files for you.

Lets see what attributes are defined in this command

archetypeGroupId

defines the GroupId in the Mule repository where the skeleton of the Mule custom policy is defined.

archetypeArtifactId

defines the ArtifactId of the archetype in the Mule repository.

archetypeVersion

A version of the archetype.

groupId

defines the GroupId for your application or module. Generally, it should be an AnypointOrganizationId where the policy will be deployed

artifactId

defines name of your policy.

We mainly work on 3 files, one is

- **XML template** where the actual logic of the policy and Mule configuration are defined.
 - Here we are logging a message using vars.msg dataweave expression, this message value should be provided by the user
 - So how does a user provide this value?
- It is done using a
- **YAML file** which renders the policy configuration UI in Anypoint API Manager.
 - If this yaml file is not provided, the policy won't be able to be applied through Anypoint API Manager.
- In pom.xml ---

Packaging type as

mule-policy , so packager plugin can successfully build the JAR,

distributionManagement section is defined pointing to user's Exchange.

mule-maven-plugin is responsible of packaging the policy into a deployable jar

maven-deploy-plugin is configured to deploy both the resulting jar and the YAML when uploading the policy to Exchange

Basics of XML template

- Within the **http-policy:proxy** element contains a **http-policy:source, this** block contains the actual logic to implement the policy.
- Any Mule Event operation that is defined before the **execute-next element** will be executed before the start of a flow with an HTTP Listener.
- The **http-policy:execute-next** element is required to continue the Mule Event processing.
 - The **http-policy:execute-next** element can trigger other **policies** or the **application** flow.
 - Any policy can stop the Mule Event processing chain by not executing http-policy:execute-next.
 - HIGHLY recommended to check out the mule documentation for more info the processing chain

<<show docs to **Understanding execution order** >>

For applying policy using API Manager

We can deploy the policy to exchange using mvn deploy

Custom API Policy – Handlebars

MCD2

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

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

```
template.xml
1 ...
2 <http-policy:proxy name="{{policyId}}-policy">
3   <http-policy:source>
4     <set-variable
5       variableName="msg"
6       value="{{message}}"/>
7
8     <logger
9       message="#{vars.msg}"/>
10
11     <http-policy:execute-next/>
12
13     {{#if upperCase}}
14       <logger
15         message="#{upper(vars.msg)}"/>
16     {{/if}}
17
18   </http-policy:source>
19 </http-policy:proxy>
20 ...
21 ...
```

```
test-policy.yaml
25 id: custom-log-message
26 name: Custom Log Message
27 description: Policy for logging
28 category: Custom
29 type: custom
30 resourceLevelSupported: true
31 encryptionSupported: false
32 standalone: true
33 requiredCharacteristics: []
34 providedCharacteristics: []
35 configuration:
36   - propertyName: message
37     name: Enter Message
38     description: Enter value to Log
39     type: string
40     optional: true
41     sensitive: false
42
43   - propertyName: upperCase
44     name: Enable Upper Case
45     description: Logs message in upper case.
46     type: boolean
47     optional: true
48     defaultValue: false
```

barahalikar.siddharth

MuleSoft Certified Developer - Level 2 (Mule 4) Preparation

Lets understand what are HandleBars,

Handlebars is a templating framework that allows users to create more complex policies.

Each policy parameter defined in a YAML Configuration file will be available as a HandleBar variable

Using handlebars we can conditionally decide which sections of the policy are applied depending on the user configuration.

- Lets assume we have aPolicy Yaml, which takes user input to Log messages in UPPER case
- If it is selected, we need to have a logic in the policy to log messages in upper case
- Here we can make use of Handle bars,
- Within the Template xml, we can use a IF condition to execute a section of policy
- This section gets executed only if the condition is true.
- If the condition is false, this section is ignored

In this way we have multiple if conditions for a more complex policy

We will be creating a custom policy to log the user/client's Location Detail.

Within the API policy, the users should define where the policy will find the Client-IP address within the request.

This is done using a dataweave expression

The policy will log the CITY, STATE, COUNTRY, CAPITAL and TIMEZONE details of the client.

Furthermore we will modify this policy using HANDLEBARS to give users an option to log either the COUNTRY NAME or all the details

5.1 Custom API Policy

- Generate skeleton
- POM Modifications



barahamercandorith

MCD2

5.2 Custom API Policy - Development

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



barahamkaranthm
barahamkaranthm

MCD2

5.3 Custom API Policy - Deployment

- Deploy to Exchange
- Configure in API Manager



barahamercandorith

MCD2

Custom API Policy - Offline

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



barahamkaranthith

Managing Offline Custom Policy – Apply & Remove

MCD2

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	<pre>mvn clean package >>> Copy the JAR >>> MULE_HOME_DIR/policies/policy-templates</pre> Create JSON file >>> MULE_HOME_DIR/policies/offline-policies
Removing Offline Custom Policies	Delete JSON file >>> MULE_HOME_DIR/policies/offline-policies



```

1 ~{
2 ~ "template" : {
3 ... "groupId" : "c738e3c8-08cf-462d-bb9a-1553ebf5008e",
4 ... "assetId" : "log-client-location-custom-policy",
5 ... "version" : "1.1.0"
6 },
7 ~ "api" : [
8 ~ ...
9 ... "id" : "17719055"
10 ...
11 ],
12 "order" : 1,
13 ~ "configuration" : {
14 ... "ipAddress" : "1.1.1.1"
15 }
16 } log-client-location-custom-policy-definition.json

```

```

1 {
2   "id" : "log-client-location-custom-policy-definition",
3   "template" : {
4     ...
5     "groupId" : "c738e3c8-08cf-462d-bb9a-1553ebf5008e",
6     "assetId" : "log-client-location-custom-policy",
7     "version" : "1.1.0"
8   },
9   "api" : [ {
10     ...
11     "id" : 17719055
12   }],
13   "online" : false,
14   "order" : 1,
15   "configuration" : {
16     ...
17     "ipAddress" : "1.1.1.1"
18   }
19 }

```

policy-definition.json

barahalikar.siddharth

MuleSoft Certified Developer - Level 2 (Mule 4) Preparation

An offline policy is applied directly to the runtime and not through API Manager.

But how do we do that?

First we need to satisfy few prerequisites like,

- The API Gateway Capabilities should be enabled in Mule Runtime Engine
- The API should be configured with a basic or proxy endpoint exists in API Manager
- The Mule application should be linked with API Autodiscovery & deployed with a HTTP(s) flow

To apply the policy,

- We need to package the policy into a deployable JAR
- Copy the JAR to MULE_HOME/Polciies/policy-tempaltes directory
- Wee need to create a JSON file in %MULE_HOME%/policies/offline-policies directory to define the configuration
 - API instance ID
 - Order of the policy in which it should be executed if we have multiple

- offline policies and
- User input Parameters

- To **remove** an offline policy, you need to delete the JSON file you created in the offline-policies directory.

5.4 Custom API Policy - Offline

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



barahatkarandhark
barahatkarandhark

MCD2

5.5 Custom API Policy - Complex

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



barahamkaranthanth

MCD2

Section 5 - Completed

MCD2

#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

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['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

Log Country Name

Log Multiple Details

barahalikar.siddharth

MuleSoft Certified Developer - Level 2 (Mule 4) Preparation

Welcome to Section 5

In this section

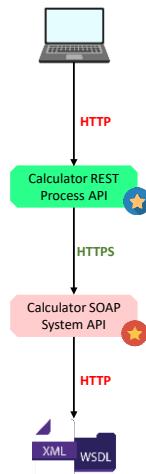
We develop a custom api poloicy from scratch and apply it using API Manager and also within Anypoint Studio

Section 6

MCD2

#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



barahalikar.siddharth

MuleSoft Certified Developer - Level 2 (Mule 4) Preparation

Section 6

In Section 6 We understand how to use TLS config for SOAP webservice and consume a service using Mutual TLS Authentication.

SOAP WebService

- Add Module
- Configure TLS Certs

PEOPLE ARE STILL USING

SOAP WEBSERVICES?

barahamercandorffh

Consume SOAP Webservice with TLS certificates

MCD2

```

1 <dependency>
2   <groupId>org.mule.connectors</groupId>
3   <artifactId>mule-wsc-connector</artifactId>
4   <version>1.6.7</version>
5   <classifier>mule-plugin</classifier>
6 </dependency>
```

```

21 <wsc:config
22   name="webServiceConsumerConfig">
23   <wsc:connection
24     wsdlLocation="http://www.dneonline.com/calculator.asmx?WSDL"
25     service="Calculator"
26     port="CalculatorSoap"
27     address="http://www.dneonline.com/calculator.asmx">
28     <wsc:custom-transport-configuration>
29       <wsc:http-transport-configuration
30         requesterConfig="httpTlsSoapConsume" />
31     </wsc:custom-transport-configuration>
32   </wsc:connection>
33 </wsc:config>
```

```

15 <http:request-config name="httpTlsSoapConsume">
16   <http:request-connection
17     protocol="HTTPS"
18     tlsContext="tlsContext" />
19 </http:request-config>
```

General Advanced Notes Help

Name: webServiceConsumerConfig

Connection

General Security Transport Advanced

Soap version: fxx SOAP11 (Default)

Mtom enabled: False (Default)

Encoding:

Connection

Wsdl location: fxx http://www.dneonline.com/calculator.asmx?WSDL

Service: fxx Calculator

Port: fxx CalculatorSoap

Address: fxx http://www.dneonline.com/calculator.asmx

```

2 <tls:context name="tlsContext">
3   <tls:trust-store
4     path="server-truststore.jks"
5     password="123456"
6     type="jks" />
7   <tls:key-store
8     type="jks"
9     path="server-keystore.jks"
10    alias="server"
11    keyPassword="123456"
12    password="123456" />
13 </tls:context>
```

barahalikar.siddharth

MuleSoft Certified Developer - Level 2 (Mule 4) Preparation

Web Service Consumer Connector consumes a SOAP web service from a Mule app to fetch data from an external source.

We can add the module to Mule app and configure the WSDL in the connector config.

This is enough for accessing the Webservice over HTTP

What if we want to consume a Webservice which has mutual authentication enabled?

Where in the connector do we mention the TLS certification for TLS communication?

Here we rely on the HTTP Request connector to configure TLS certs and use that configuration within Webservice connector

Secure Data in Transit

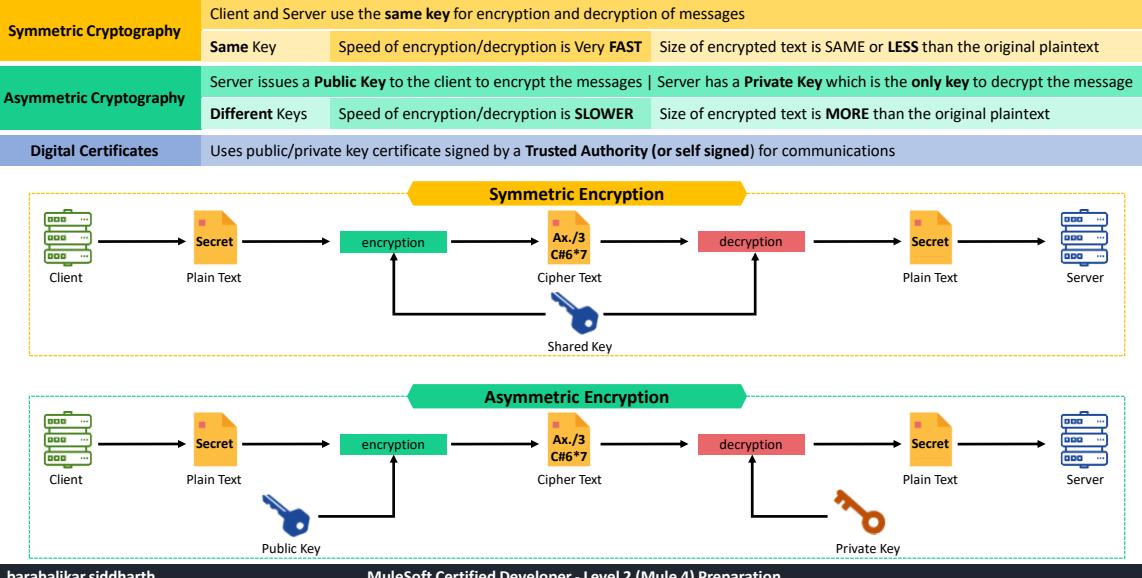
- Secure Communication – Using Cryptography
- Digital Certificates
- One-way TLS
- Two-way TLS
- Create Keys & Certificates ([JKS](#)) 2-Way-TLS
- Create Keys & Certificates ([PKCS12](#)) 2-Way-TLS



barahmierandmehr

Secure Communication – Using Cryptography

MCD2



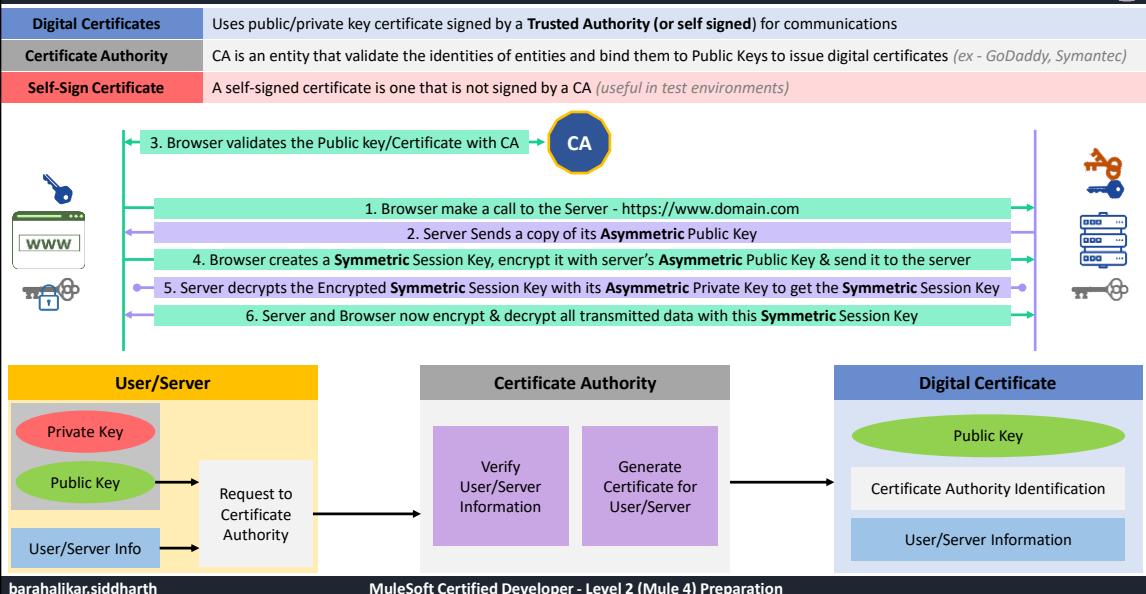
barahalikar.siddharth

MuleSoft Certified Developer - Level 2 (Mule 4) Preparation

Within secure communication, we understood the differences between Symmetric and Asymmetric encryption

Digital Certificates

MCD2



Here we understood what are digital certificates and these are obtained by the server/user

We also saw the interaction between a client and server exchanging keys to secure the data transmission

One-way TLS

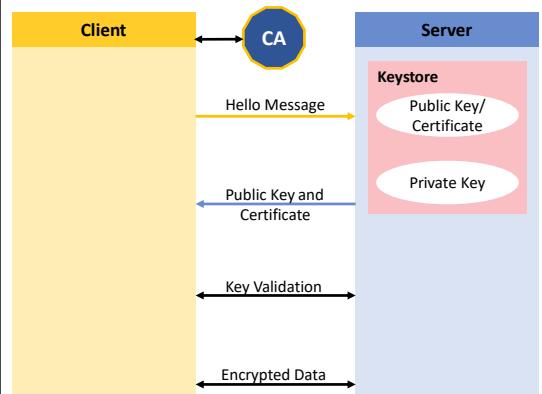
MCD2

Keystore Is a repositories that contains **Public certificates**, plus the corresponding **Private key** for clients or servers.

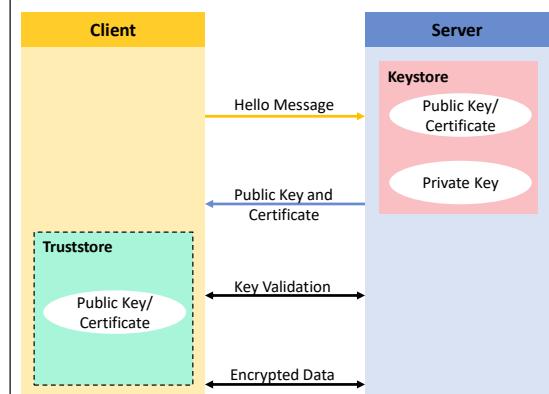
Truststore Contains trusted certificates on a TLS client used to **validate** a TLS Server's Public certificate presented to the client (*typically self-signed certificates*)

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

One-way TLS (no Truststore)



One-way TLS (optional Truststore)



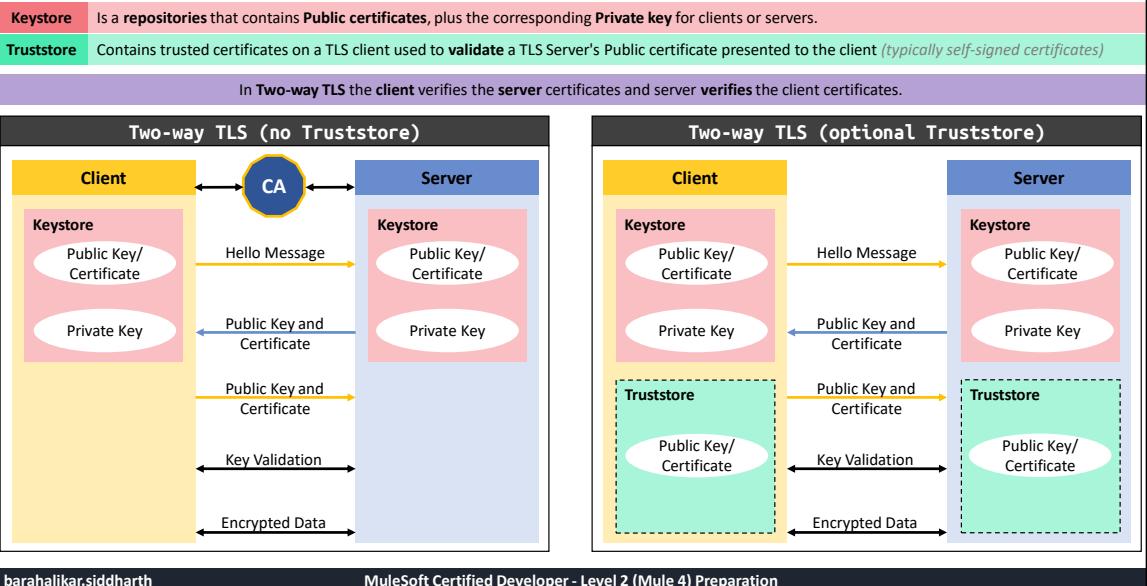
barahalikar.siddharth

MuleSoft Certified Developer - Level 2 (Mule 4) Preparation

Then we talked about one way tls and how it works with and without a truststore

Two-way TLS

MCD2



barahalikar.siddharth

MuleSoft Certified Developer - Level 2 (Mule 4) Preparation

Finally we talked about 2 way tls and how it works with and without a truststore.

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
```

barahalikar.siddharth

MuleSoft Certified Developer - Level 2 (Mule 4) Preparation

We also talked about JKS vs PCKS formats and generated a keystore which was used for 1 way tls.

Here we will see how to generate jeystore and trsutsores using JKS format

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



barahmierandmehr

MCD2

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



barahmierandmehr

MCD2

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



barahmierandmihir

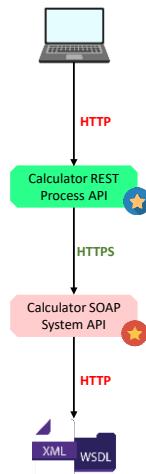
MCD2

Section 6 - Completed

MCD2

#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



barahalikar.siddharth

MuleSoft Certified Developer - Level 2 (Mule 4) Preparation

Section 6

In Section 6 We understand how to use TLS config for SOAP webservice and consume a service using Mutual TLS Authentication.

Section 7

MCD2

#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

barahalikar.siddharth

MuleSoft Certified Developer - Level 2 (Mule 4) Preparation

Welcome to Section 7

In this section we will work on Munit testing.

We will start by configuring Munit test suite for a Mule application

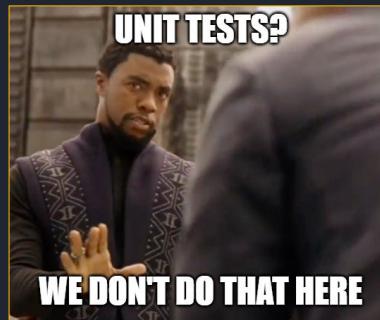
Write Munit test cases by

- Setting the event data
- Mocking dependencies
- Asserting the response
- Verifying the calls and processors data
- Refactor Munit code and
- Test flows with custom error handler

We will end this section by automating Munit using Munit recorder

MUnit Testing

- MUnit Basics



barahimkarandanth

MUnit Introduction

The screenshot shows the Anypoint Studio interface. On the left, there's a tree view with nodes like 'Mule', 'Manage Dependencies', 'AnyPoint Platform', 'MUnit' (which is expanded), and 'Run As'. A context menu is open over the 'MUnit' node, with options 'Run Tests' and 'Configure MUnit Maven Support'. Below the tree, a code editor displays a portion of a Maven pom.xml file. Several dependencies are highlighted with yellow boxes, specifically those related to the munit-maven-plugin, munit-runner, mule-plugin, assertions, and org.mule.weave modules.

This is a screenshot of the 'https-test-suite-httpsFlowTest' configuration dialog. It has tabs for 'Execution' and 'Validation'. Under 'Execution', it says 'Run flows in here' and 'Add assertions and verifies here'. There are two buttons at the bottom: 'UNIT TEST' and 'INTEGRATION TEST'. The 'UNIT TEST' button is selected.

This part of the dialog shows basic settings for the test. It includes fields for 'Name' (https-test-suite-httpsFlowTest), 'Basic Settings', 'Scenario description' (Test Response is not null), 'Ignore test' (fx, set to False (Default)), 'Tags' (unit_test,integration_test), and 'Test timeout (ms)' (120000).

Resource Coverage - #Processors 3 / 3

100%

Required Resource Coverage : N/A

Required Container Coverage : N/A

Containers

Name	Type	#Covered Processors	#Processors	Coverage*
httpsBinlow	Flow	3	3	100%

barahalikar.siddharth

MuleSoft Certified Developer - Level 2 (Mule 4) Preparation

The main objective of unit testing is **to isolate written code to test and determine if it works as intended.**

Unit tests help to find problems early in the development and fix them quickly

- MUnit is a Mule application testing framework which allows you to build automated tests for your Mule integrations and API's.
- MUnit is very well integrated with Anypoint Studio.
- Each Munit test is essentially a Mule flow
- Tests are executed in a Mule runtime and reports are generated
- We can configure Munit maven support for each application,
- It modifies the application pom to include munit dependencies like munit-runner, munit-tools, & assertions module
- A Munit ttest flow is divided in three scopes and all are optional.
 - the Behavior scope is ■ where all preconditional before execution are set, like mocks, spys
 - The Execution scope ■ is where we have testing logic, references to the Mule flow and setting event data
 - The Validation scope ■ is the place where we do assertions

- Each test flow has multiple configurations options like,
 - tags which can be used in maven commands
 - Timeouts and the option to
 - Ignore the tests etc
- Finally it also gives a detailed coverage report on the mule flows and its processors being tested.

MUnit Operations

MCD2

Set Event	Assert That	Mock When
 Set Event <p>allows you to define a Mule Event</p> <ul style="list-style-type: none"> • Set Payload • Set Attributes • Set Errors • Set Variables 	 Assert that <p>allows you to run assertions to validate the Mule event after the production code runs</p> <ul style="list-style-type: none"> • #[MunitTools::equalTo('example')] • #[MunitTools::nullValue()] 	 Mock when <p>allows you to mock any processor and return a predefined result instead of executing the processor itself</p> <ul style="list-style-type: none"> • Mocking Using Then-Return • Mocking Using Then-Call • Mocking Errors
Verify Call	Spy	Flow Error/Exception Tests
 Verify call <p>you can validate if a specific processor has been called with a particular set of attributes a specific number of times</p> <ul style="list-style-type: none"> • times=2 • atLeast=2 • atMost=4 	 Spy <p>allows you to spy & assert what happens before and after an event processor is called</p>	<pre> 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> </pre> <p>Tests can be configured to expect an exception thrown.</p> <p>Mock when allows to simulate error conditions in tests.</p>

Within the Munit operations, we will be exploring the following operations.

We will use set event data operation to define the mule event like

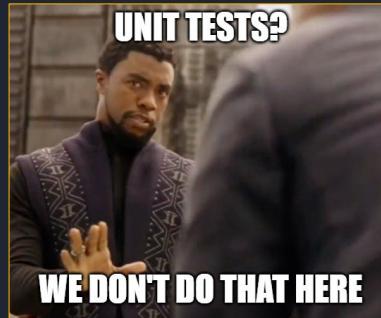
- Set Payload
- Set Attributes
- Set Errors
- Set Variables

We will use assert that operation which allows you to run assertions to validate the Mule event after the code run

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

7.1 MUnit Testing

- Import new app
- Deploy & Test

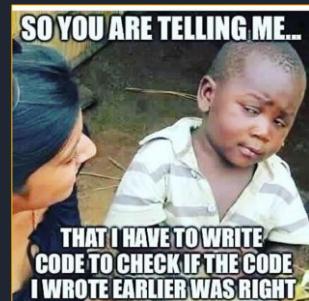


barahimkarandanjah

MCD2

7.2 MUnit Testing

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

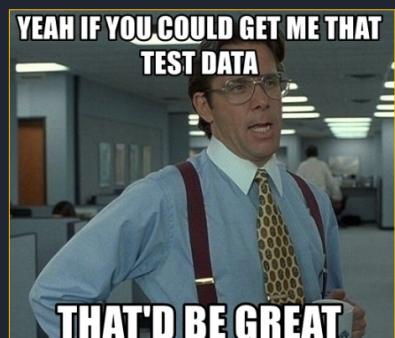


barahmierandmuth

MCD2

7.3 MUnit Testing

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

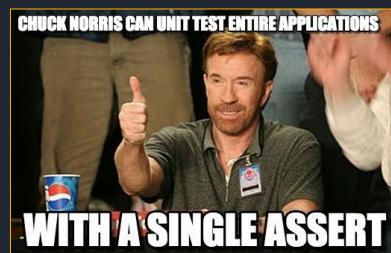


barahamcarlsmith

MCD2

7.4 MUnit Testing

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



barahamcaronduffy

MCD2

7.5 MUnit Testing

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

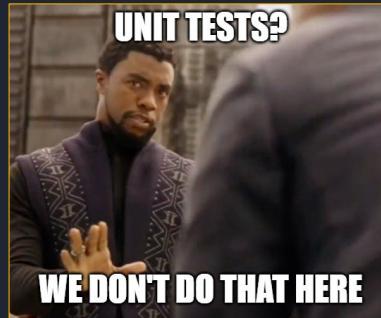


barahierachy

MCD2

7.6 MUnit Testing

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



barahimkarandanjah

MCD2

7.7 MUnit Testing

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



barahmierandmehr

MCD2

7.8 MUnit Testing

- Refactoring Munit Tests and Data



barahmcarlsmith

MCD2

7.9 MUnit Testing

- Tests Flow with custom error handler
- Mock Errors



barahamcaronduh

MCD2

MUnit Testing

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

TESTS CAN'T FAIL



IF YOU DON'T RUN ANY

barahamkaranthanth

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

MCD2

<code>mvn clean test</code>	<code>mvn clean package -DskipTests</code>	<code>mvn clean test -Dmunit.tags=<munit-tag></code>		
		<code>mvn clean test -Dmunit.test=<regex-test-suite></code>		
		<code>mvn clean test -Dmunit.test=<regex-test-suite>#<regex-test-name></code>		
MULE or MULE_EE				
runtimeProduct	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%				
Does not count as coverage data.				
Ignoring a flow or a file	Does not cause a build to fail if the flow is not tested or if the flow does not reach coverage metrics			

barahalikar.siddharth MuleSoft Certified Developer - Level 2 (Mule 4) Preparation

MUnit provides a Maven plugin that enables you to run your MUnit tests as part of your continuous integration environment.

This is a sample Munit maven pom configuration

Talking about Coverage

These configurations only apply when you execute your MUnit tests using the Maven plugin.

They do not apply when running tests from Studio.

One of the features of MUnit Coverage is to fail the build if a certain coverage level is not reached.

- MUnit Coverage handles three different levels:
 - **Application** level coverage of all your application.
 - **Resource** level coverage of each individual Mule configuration file.
 - **Flow**: level coverage of event processors in each flow.
- Fail Build - if **false**, and the coverage levels are not reached, the build will not fail, but will give a warning

The reports can be generated in 4 formats >> console, html, json and sonar

>>

- Another feature is the ability to ignore either a flow or a file.
- This way, the ignored resource:
 - Does not count as coverage data. And it also
 - Does not cause a build to fail if the flow is not tested or if the flow does not reach coverage metrics

You can specify the type of runtime in which your applications being tested will run.

We add the runtimeProduct tag , it could e:

- MULE for community edition. But
- if app has enterprise connectors/config we need to use MULE_EE for MUnit testing (*example – Mule Secure Properties Config*) we use MULE_EE for enterprise edition.

To test we can use these cmd's

Mvn test for testing

-DskipTest to skip testing

For –Dmunit.tags - Only tests tagged with these names will run.

We can have multiple test suite in the app,

We can select running all tests defined in the test suites or

We can also mention the test name which should be executed in the test suite

There is a lot more to Munit Maven Plugin, please refer the documentation page for more info

7.10 MUnit Testing

- Maven MUnit Tests
 - Failure

TESTS CAN'T FAIL



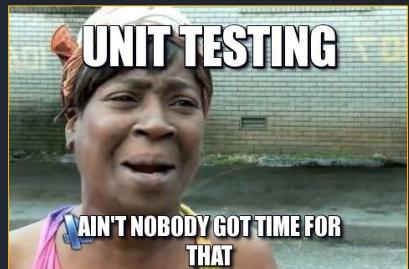
IF YOU DON'T RUN ANY

barahamercardorath

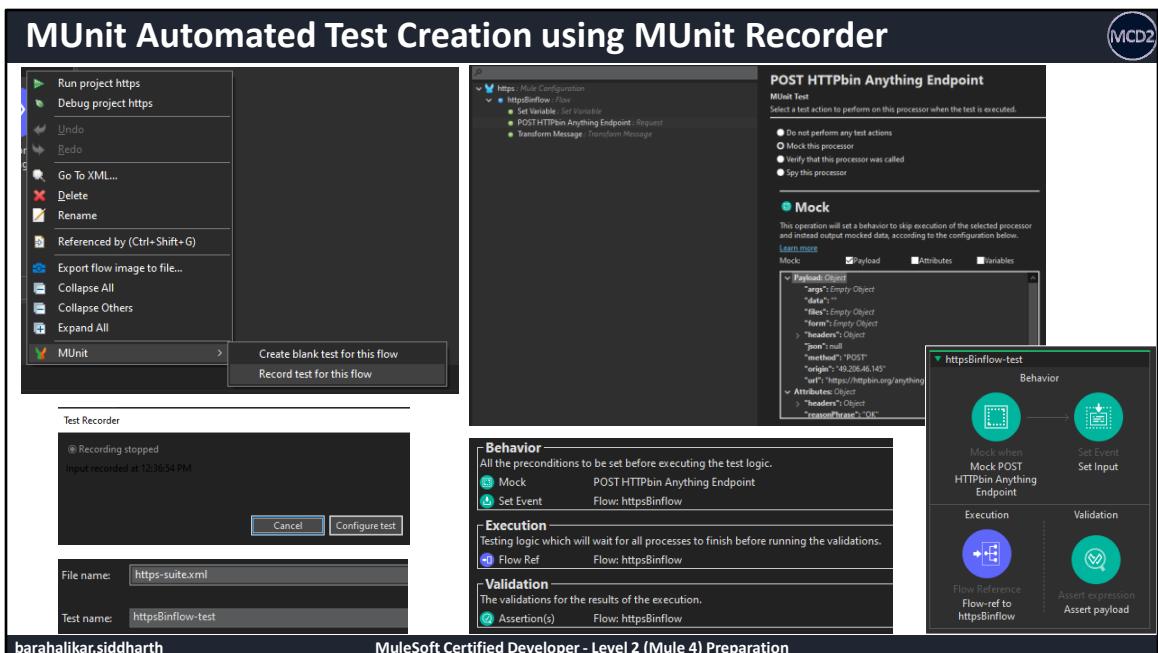
MCD2

MUnit Testing

- Automated MUnit Recorder
- Configuration



barahamercanthanh



MUnit Test Recorder is a tool to automate the creation of **Munit test cases** in MuleSoft.

The test recorder enables you to record a processing flow and then configure a unit test based on the captured event.

The test recorder captures real data as it goes through your application in Anypoint Studio.

By capturing the flow execution, MUnit can automatically generate an MUnit test so that you can configure the necessary mocks and assertions for your test.

Steps to achieve this are,

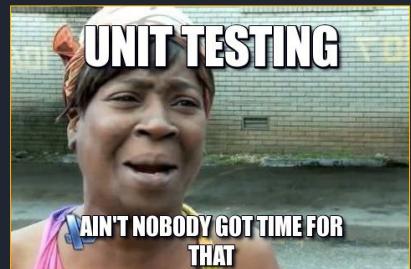
- We can right click on any flow and select Record test for this flow
- It will deploy the app and wait for the request call.
- Once the flow gets executed, we can configure the test
- It will capture everything from the request and response like payload, attributes and variable
- We can decide which components needs to mocked/verified/spied based on the

requirement and select the options

- It is going to automatically generate the tests based in our input.
- It has few Limitations –
- You cannot create tests for flows with Mule errors raised inside the flow
- The recorder does not support mocking a message before or inside a Foreach processor.
- For more information please refer Mule documentation

7.11 MUnit Testing

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



barahamercanthanh

MCD2

Section 7 – Completed

MCD2

#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

barahalikar.siddharth

MuleSoft Certified Developer - Level 2 (Mule 4) Preparation

Welcome to Section 7

In this section we will work on Munit testing.

We will start by configuring Munit test suite for a Mule application

Write Munit test cases by

- Setting the event data
- Mocking dependencies
- Asserting the response
- Verifying the calls and processors data
- Refactor Munit code and
- Test flows with custom error handler

We will end this section by automating Munit using Munit recorder

HTTPS API Proxies

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



barahamercardonth

(MCD2)

Build HTTPS API Proxies

MCD2

The diagram illustrates four scenarios for building HTTPS API proxies:

- Scenario 1:** A user sends a request to <http://localhost:8081/proxy/nuclear/{code}>. The API Proxy forwards the request to the App Implementation at <https://localhost:7777/api/nuclear/{code}>. The response is {"power_plant": "KAPS", "capacity": "700MW", "location": "Gujarat", "operational": "yes"}.
- Scenario 2:** A user sends a request to <http://localhost:8081/proxy/nuclear/{code}>. The API Proxy forwards the request to the App Implementation at <https://localhost:7777/api/nuclear/{code}>. An error message is displayed: `javax.net.ssl.SSLHandshakeException:PKIX path building failed : unable to find valid certification path to target'`.
- Scenario 3:** A user sends a request to <http://localhost:8081/proxy/nuclear/{code}>. The API Proxy forwards the request to the App Implementation at <https://localhost:7777/api/nuclear/{code}>. The response is {"power_plant": "KAPS", "capacity": "700MW", "location": "Gujarat", "operational": "yes"}.
- Scenario 4:** A user sends a request to <https://localhost:8082/proxy/nuclear/{code}>. The API Proxy forwards the request to the App Implementation at <https://localhost:7777/api/nuclear/{code}>. The response is {"power_plant": "KAPS", "capacity": "700MW", "location": "Gujarat", "operational": "yes"}.

On the right side, there are four terminal command examples corresponding to the scenarios:

- Scenario 1:** `keytool -genkey -alias app \ -keysize 4096 -keyalg RSA \ -keystore app-keystore.jks`
- Scenario 2:** `keytool -export -alias app \ -keystore app-keystore.jks \ -file app_public.crt`
- Scenario 3:** `keytool -import -alias proxy \ -keystore proxy-truststore.jks \ -file app_public.crt`
- Scenario 4:** `keytool -genkey -alias proxy \ -keysize 4096 -keyalg RSA \ -keystore proxy-keystore.jks`

Below the diagrams, there are two screenshots of the MuleSoft Anypoint Platform interface:

- Secret Groups:** Shows a list of secret groups, including "Create Secret Group", "Name", and "api-proxy-secret-group". Arrows point from "Keystore" and "Truststore" fields to the "TLS Context" field.
- Edit TLS Context:** Shows the configuration for "api-proxy-tls-context". It includes fields for "Name" (api-proxy-tls-context), "Target" (Mule), "TLS Version" (TLS 1.2 selected), "Keystore" (api-proxy-keystore), and "Truststore (Optional)" (api-proxy-truststore). A "Select TLS Context" dialog is open, showing "Secret group" (api-proxy-secret-group) and "TLS Context" (api-proxy-tls-context).

<https://self-signed.badssl.com/>

Create proxy >> add url

Deploy

Get error ssl engine failure

Export .cer >> add to truststore

Add truststore to Secret group >>> create tls

Redeploy >> test

Generate a keystore >> add to secret group

CREATE ANOTHER TLS to use KEYSTORE

Redeploy >> test

Use on-prem to deploy app and proxy api

Add server to runtime manager

App >> nuclear app >> one flow >> one https listener with KEYSTORE >> port

anything other than 8082/8081 >> path /api/code

Proxy >> basePath /proxy >>
>> without tls
>> with outbound tls
>> plus inbound tls

Secret Manager >> one for truststore >> one for keystore

A-1 HTTPS API Proxies

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



barahimcarthorpe

MCD2