* What are the different types of flow
* **Subflow** – A subflow is always synchronous. It is similar to a synchronous flow, subflow executes in the same thread of the calling process. Calling process triggers the sub-flow and waits for it to complete and resumes once the sub-flow has completed. Subflow won’t have exception handling strategy.
* **Synchronous Flow**– Same as sub-flow, the only difference is that in synchronous flows you need to separately define an exception strategy to it, it does not inherit the exception strategy of its calling flow.
* **Asynchronous Flow** – As in sub-flow and synchronous flow, calling process triggers the sub-flow and waits for it to complete; for asynchronous flow the flow, calling process triggers an asynchronous flow and moves ahead to its next activity. An asynchronous flow executes in parallel to its calling/parent flow in a different thread. An asynchronous flow does not return its output it its parent/calling flow.
* **Private Flow** – flow that does not have an inbound connector in the source. Means a private flow cannot start of its own on receiving the inbound message as it does not have any inbound connector, A private flow can only be called using flow-ref same as sub-flow.
  + Private flows are same as standard flows without the message source. They can also be called using the flow-reference element.
  + Private flows are different from subflows in terms of Threading and Exception Handling. They have their own Exception Handling. The exception will be trapped by the local Exception Handling and it will not be propagated to the main flow. It means that if an exception occurs in a Private flow and it is handled properly when the call goes back to the main calling flow, the next message processors continue executing.
  + Private flows also receive the same message structure from the main calling flow. But Private flows create a new execution context.
  + **subflow is always recommended, except when the exception needs to be handled in a separate exception handling flow different from the main flow or if you are implementing an asynchronous pattern.**
* Processing Strategies in Mule4

Mule 4 has no flow processing strategies. Mule improved execution engine which takes care of processing strategies for your mule application also for the better performance.

Mule 4 execution engine follows the non-blocking architecture which means the engine invokes the API and does not wait for a response by blocking the threads. The API can take more time than expected but the threads are going useless. To maximise the usage of the thread’s engine will make use of those threads to finish some other operations whenever the response comes then the engine will listen and return to that call.

**What about Mule 3?**

* Synchronous Flow Processing Strategy
* Queued Flow Processing Strategy
* Asynchronous Flow Processing Strategy
* Thread Per Processing Strategy
* Queued Asynchronous Flow Processing Strategy
* Non-blocking Flow Processing Strategy
* Custom Processing Strategy
* Mule 4 Variables

Variables are used to store the event values which we can use with in the flow. We can store the current message (using the **message** keyword), the current message payload (using the **payload** keyword) or just the current message attributes (using **attributes** keyword).

**You can create or update variables in these ways:**

* Using the Set Variable component.
* Using a Target Variable from within an operation, such as the Read operation to the File connector or the Store operation to the Database connector.
* Using the DataWeave Transform Component.
* Using Scripting Component (in scripting module)

**How to access variables in your flow?** #[vars.varName]

What about mule 3?

* 1. Flow variable
  2. Session variable
  3. Record variable
* What are Shared Resources in Mule and how are they been used?

We can make connectors as reusable component by defining them as common resources and expose them to all applications deployed under a same domain, these resources are known as shared resources. These shared resource needs to be defined inside Mule Domain Project and then referred to each of the projects that are meant to use the elements in it.

* How to call Java in Mule?

By using the **invoke** component we call Java methods (either instance or static). Their return value is placed in the payload of the output message or can be placed in a target variable.

* How can we create and consume SOAP service in Mule?
* Creating SOAP Service – We can create a SOAP service same as we create Mule Project With RAML, the only change is instead of RAML we need to import Concert WSDL.
* Consuming SOAP Service – We can use **Web Service Consumer** to consume the SOAP service
* How do you get authorization values in your flow?

Any frequently modified values and connection authorization values we keep in properties file. We call those values in connectors. In future if there is any change in configuration values, we don’t need to change in all connectors, we just need to modify in properties file.

* How to call data from property file?
* Using ${} - "${Jane}"
* Using p() - p("Jane")

**Differences in ${} and p()**

* ${} can be used in both MEL and DataWeave while p() is limited to DataWeave
* p() can take flow variable as argument. p(flowVars.name)
* Argument for p() can be derived from concatenation also, but not for ${}. p(flowVars.name ++ "lastname")
* How to encode your credentials from properties file? In lieu of hard code
  + Go to your Mule app configuration file.
  + Select Global Elements tab.
  + Click Create.
  + Select Secure Properties Config.
  + Configure the global element with a File location, Key, Algorithm, Mode, use random IV, File level encryption, and Encoding.
  + Click OK.
* Can you explain API Life cycle?

The five stages of an API lifecycle are:

* + Planning and Designing the API
  + Developing the API
  + Testing the API
  + Deploying the API
  + Retiring the API (Versions)

**Retiring the API**:- Deprecation is a natural part of the API lifecycle. It is the phase where support for an API’s version, or in many cases, an entire API itself, is discontinued. organizations to indicate a deprecated API that’s not for consumption, so end consumers can know exactly which API is stable, and which isn’t.

* Any design patterns you worked on?

Yes, we have implemented multiple design patterns based on the requirement.

* Publish Subscribe pattern:-

It has one input channel that splits into multiple output channels, one for each subscriber. When an event is published into the channel, the Publish-Subscribe Channel delivers a copy of the message to each of the output channels

* Point-to-Point integration pattern:-

A Point-to-Point Channel ensures that only one receiver consumes any given message. If the channel has multiple receivers, only one of them can successfully consume a particular message. If multiple receivers try to consume a single message, the channel ensures that only one of them succeeds

* Guaranteed Delivery Message pattern:-

With Guaranteed Delivery, the messaging system uses a built-in data store to persist messages. Each computer the messaging system is installed on has its own data store so that the messages can be stored locally. When the sender sends a message, the send operation does not complete successfully until the message is safely stored in the sender’s data store. Subsequently, the message is not deleted from one data store until it is successfully forwarded to and stored in the next data store.

* Invalid Message Channel: -

The receiver should move the improper message to an Invalid Message Channel, a special channel for messages that could not be processed by their receivers.

* Dead Letter Channel: -

When a messaging system determines that it cannot or should not deliver a message, it may elect to move the message to a Dead Letter Channel.

* WebService Synchronous Consumer pattern
* File based integration pattern
* JDBC integration pattern
* Batch processing and scheduled job integration pattern
* What is mule Requester or Requester Module?

This module enables the request of a resource at any point in a flow. This resource can be a file, a message (from VM, JMS, AMQP, etc.), an e-mail, etc. It's intended for resources that originally can only requested by message sources.

**Some of its common use cases are:**

* Load a file in the middle of a flow for processing.
* Consume messages (one, N, all) from a queue in the middle of a flow.
* Pull messages from a mail server on demand, to use its data in an enricher for example.
* What is the use of parsed template?

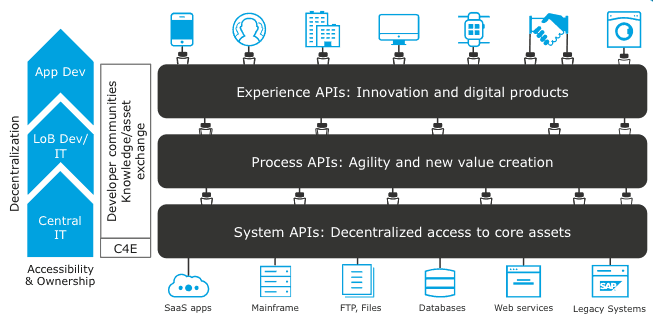
Parse Template is the Mule component to use for processing a template and obtaining a result. A template is defined as text with embedded Mule expressions that are evaluated and replaced with their result.

You can configure template through an external file reference, or you can embed it in the component definition itself.

* Can you brief about API led connectivity?

Describe about the layers in mule

* + **System APIs** – these usually access the core systems of record and provide a means of insulating the user from the complexity or any changes to the underlying systems. Once built, many users, can access data without any need to learn the underlying systems and can reuse these APIs in multiple projects.
  + **Process APIs** – These APIs interact with and shape data within a single system or across systems (breaking down data silos) and are created here without a dependence on the source systems from which that data originates, as well as the target channels through which that data is delivered.
  + **Experience APIs** – Experience APIs are the means by which data can be reconfigured so that it is most easily consumed by its intended audience, all from a common data source, rather than setting up separate point-to-point integrations for each channel. An Experience API is usually created with API-first design principles where the API is designed for the specific user experience in mind.



* Application Networks

Set of applications to connecting to set of applications, so that they can exchange the data through Api’s.

* What is API Autodiscovery?

API Autodiscovery is a mechanism that manages an API from API Manager by pairing the deployed application to an API created on the platform. API Management includes tracking, enforcing policies if you apply any, and reporting API analytics. Autodiscovery process is identifying the API by providing the API name and version.

* + 1. We configure the client id and client secret from the respective env by goint to Anypoint platform->Credentials -> Api Manager(validate)
    2. You need to capture the API Id from the API Manager and go to global configuration and enter API ID and select the flow.
* Error Handling in Mule 4

There are 3 types of error handling mechanisms in Mule 4

* On-Error Continue
* On-Error Propagate
* Try Catch Scope

**On-Error Continue** catches the error and does not report it as an error, thus the processing of the flow continues even after the error has occurred. This error handler can be used in flows where you don’t want to stop the flow processing even if an error has occurred.

**On-Error Propagate** works exactly as the Mule 3 Catch exception strategy. In case of any errors, On-Error Propagate processes the error message and re-throws the error to its parent flow. No further processing is done on that particular flow.

**Try-catch scope** can be used within a flow to do error handling of just inner components. Try-catch scope can be very useful in cases where we want to add a separate error processing strategy for various components in the flow.

* How do you handle errors Globally? (Global Error Handling)

For every project there should be common errors and exceptions which needs to be handled globally. So, that we can point common errors to the global configuration.

* Can we call a flow from another project?

Yes, it is possible... You can do it in 2 ways: -

* Create a domain project and define a Shared VM connector in the domain project, which will be used to call flows of another project under the domain.
* Another way is to create a Jar file of the application whose flow need to be called and then add the jar file as a dependency with the project and import the flow to be call in Spring bean. Flow-ref can be used here to call the flow.
* What is Scatter/Gather
* The routing message processor Scatter-Gather sends a request message to multiple targets concurrently. It collects the responses from all routes and aggregates them into a single message.
* The Scatter-Gather router sends a message for concurrent processing to all configured routes. The thread executing the flow that owns the router waits until all routes complete or time out.
* If there are no failures, Mule aggregates the results from each of the routes into a message collection (MessageCollection class). Failure in one route does not stop the Scatter-Gather from sending messages to its other configured routes, so it is possible that many, or all routes may fail concurrently.

By default, if any route fails, Scatter-Gather performs the following actions:

* Sets the exception payload accordingly for each route
* Throws a CompositeRoutingException, which maps each exception to its corresponding route using a sequential route ID
* Catching the CompositeRoutingException allows you to gather information on all routes, whether successful or failed.
* What is the difference between ESB and JMS ?
* ESB provides the middleware and interfaces that allow businesses to connect their applications without writing code.
* JMS provides messaging capability and facilitates communication between the modules/applications.
* Cache Scope

The Cache scope is for storing and reusing frequently called data. You can use a Cache scope to reduce the processing load on the Mule instance and to increase the speed of message processing within a flow. It is particularly effective for these tasks:

* Processing repeated requests for the same information.
* Processing requests for information that involve large, non-consumable payloads.

Create global configuration by Reference an ObjectStore or create one for your Cache component to reference.

**Select a mechanism for generating a key used for storing events within the Caching Strategy**:

* Default
* Key Expression: A DataWeave expression (for example, keyGenerationExpression="#[vars.requestId]").

Note that for two requests that are the same ("equal"), you need to generate the same key. Otherwise, you can get wrong results.

* Key Generator: Requires you to implement the com.mulesoft.mule.runtime.cache.api.key.MuleEventKeyGenerator interface.
* Validation Module in Mule
* The Validation module provides an easy out-of-the-box way to verify that the content of a message in your flow matches a given set of criteria.
* The main advantage of the Validation Module over using filters is traceability, as filters all raise identical exceptions, making it hard for us to know where the exception was caused. So if we have two filters in the same flow, we cannot know which one failed and we cannot customize the exception type. Validators, on the other hand, raise an exception with a meaningful message attached, so we can customize the exception type.

The validations module was designed following these principles:

* If the message doesn’t meet the specified criteria, the validation fails and a ValidationException is thrown.
* By default, this exception has a meaningful message attached. You can optionally customize this message and change the type of exception thrown, as long as the new exception type has a constructor that overrides Exception(String).
* In case you want to throw an Exception type that lacks this constructor, or in which its creation is not trivial, or in which you want it to contain additional information.
* **How to Use Validator**

There are two ways of using a validator: through a Message Processor, or through a MEL function.

* **Email Address Validator:**

<validation:is-email email=”#[json:email]” />

* **Regular Expression Validator**

<validation:matches-regex value=”#[json:Name]” regex=”/^[A-z]+$/” message=”Name can not contain Integer value” />

* **Not Empty check Validator**

<validation:is-not-empty value=”#[json:employeeId]” message=”Employee Id is mandatory to supply” />

* **Size Validator**

<validation:validate-size value=”#[json:age]” min=”2″ max=”3″ message=”Please provide correct age” />

* **Not Null Validator**

<validation:not-null expression=”#[value]” value=”#[payload]” />

* **Number Validator**

<validation:is-number value=”#[json:Id]” numberType=”INTEGER” message=”ID can not be String or any other data Type” />

**Below are the list of validations available in Validation Module:**

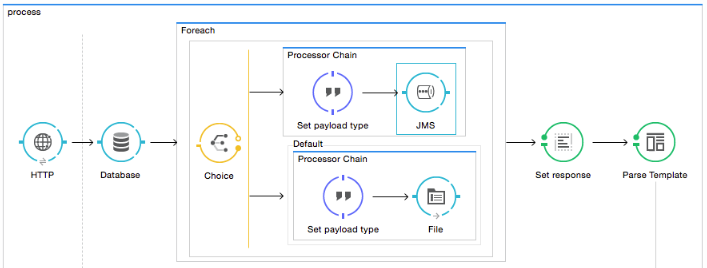
* Is Ip
* Is URL
* Is blank string
* Is elapsed
* Is email
* Is empty collection
* Is false
* Is not blacklisted ip
* Is not blank string
* Is not empty collection
* Is not null
* Is null
* Is number
* Is time
* Is true
* Matches regex
* All
* Any
* How For Each Scope works?

The For Each scope is one of the scopes available in Mule, which splits a collection into individual elements and processes them iteratively through the processors in the scope, then returns the original message to the flow. This scope basically simplifies the splitting and aggregation of message collections.

* The For Each scope does not change the type of the message collection and does not change the contents of the of the original message collection, if the individual collection elements are immutable. That is, after For Each splits a message collection and processes the individual elements, it does not re-aggregate those individual elements into a MuleMessageCollection; rather, it returns the original message
* The For Each scope is versatile; it can iteratively process elements from any type of collection, including maps, lists, arrays, and MuleMessageCollection.

## IMP Points of For Each:

* You can insert any message processor, except inbound connector, in the For Each scope. If we drag a two-way connector into a For Each scope, Mule automatically converts it to an outbound-only connector.
* When the For Each scope receives a message payload that is not a collection type like maps, lists, arrays, and MuleMessageCollection, it throws an IllegalArgumentException.
* The rootMessage variable (flow variable), which is associated with the message by default, contains a reference of the complete, unsplit message collection.
* The counter variable (flow variable) is associated with the message by default. For Each uses a counter variable to record the number of the elements it has processed.



* What is the use of Until success flow control?

The until-successful scope processes messages through the processors within its scope until the operation succeeds. Until-successful’s processing occurs asynchronously from the main flow. After passing a message into the until-successful scope, the main flow immediately regains control of the thread.

* What is the use of Round Robin flow control?

The Round Robin router iterates through a list of two or more routes in order, but it only routes to one of the routes each time it is executed. It keeps track of the previously selected route and never selects the same route consecutively.

For example, the first time Round Robin executes, it selects the first route. The next time, it selects the second route. If the previously selected route is the last route in the list, Round Robin jumps to the first route.

* How can you implement and use of first successful?

The First Successful router iterates through its list of child routes, each containing a set of components, until the first successful execution. If none succeed, an exception is thrown.

* Data Weave:

DataWeave is the MuleSoft expression language for accessing and transforming data that travels through a Mule app. DataWeave is tightly integrated with the Mule runtime engine, which runs the scripts and expressions in your Mule app.

* What is map?

Map is used to iterate through an array.  Transforms items from the given list (array) into a new list using the specified mapper function.

* What is mapObject?

Iterates over an object using a mapper that acts on keys, values, or indices of that object

* What is flatten?

It is used to combine multiple arrays as single array

* Can we call any flow from Data Weave?

This **lookup** function enables you to execute a flow within a Mule app and retrieve the resulting payload.

the lookup function enables you to execute another flow within your app and to retrieve the resulting payload. It takes the flow’s name and an input payload as parameters. For example**, lookup("anotherFlow", payload)** executes a flow named anotherFlow.

* How to add conditions in DataWeave to filter the xml data

Use filter function to filter the data with specific condition.

* How to handle null values from Data Weave?

**%output application/xml skipNullOn="everywhere"**

Which can be set to 3 different values: elements, attributes and everywhere.

* how to remove duplicates in dataweave

we remove duplicate by using **distinctBy**

* In Data Weave can we call any Java methods?

Yes we can do it in below ways.

* Import the class and then refer to the method:
* Import one or more methods instead of the whole class
* If the method is a static method, import and call it in a single line
* [Security Schemes](https://github.com/raml-org/raml-spec/blob/master/versions/raml-10/raml-10.md#security-schemes)

**Security Schemes available from RAML level declaration**

* [Basic Authentication](https://github.com/raml-org/raml-spec/blob/master/versions/raml-10/raml-10.md#basic-authentication)
* [OAuth 1.0](https://github.com/raml-org/raml-spec/blob/master/versions/raml-10/raml-10.md#oauth-10)
* [OAuth 2.0](https://github.com/raml-org/raml-spec/blob/master/versions/raml-10/raml-10.md#oauth-20)
* [Digest Authentication](https://github.com/raml-org/raml-spec/blob/master/versions/raml-10/raml-10.md#digest-authentication)
* [Pass Through](https://github.com/raml-org/raml-spec/blob/master/versions/raml-10/raml-10.md#pass-through)
* [x-<other>](https://github.com/raml-org/raml-spec/blob/master/versions/raml-10/raml-10.md#x-other)

[**Basic Authentication**](https://github.com/raml-org/raml-spec/blob/master/versions/raml-10/raml-10.md#basic-authentication)**:**

Basic Authentication we don’t need to specify any settings in API Definition.

#%RAML 1.0

title: Dropbox API

version: 1

baseUri: https://api.dropbox.com/{version}

securitySchemes:

basic:

description: |

This API supports Basic Authentication.

type: Basic Authentication