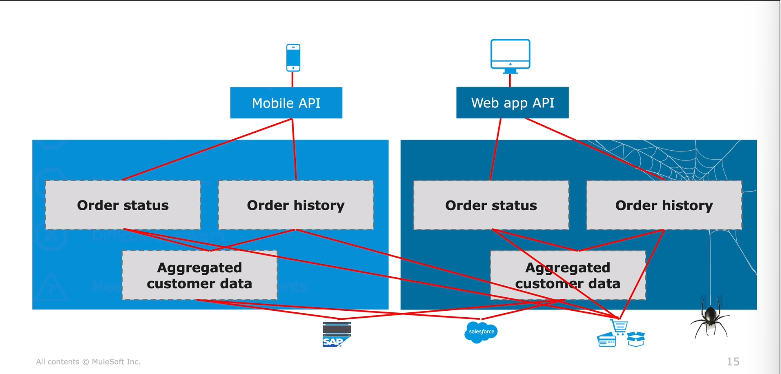
*CERTIFICATION COURSE NOTES*

* API :

It takes same time to build a mobile application and a web application. Even though the functionality is same for both web and mobile apps, we can't create the mobile app faster by using the existing web application.

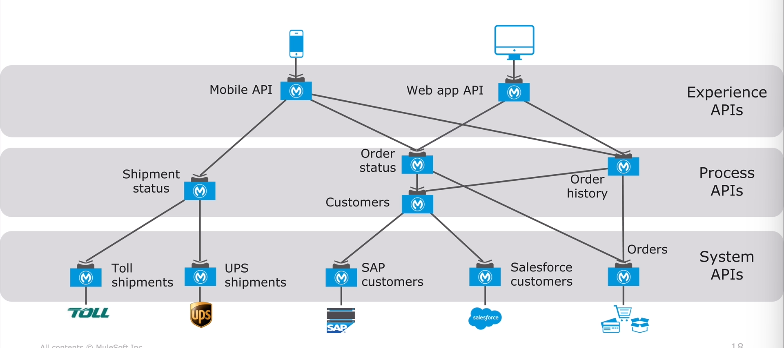


Apps are built using compassable APIs and in order to fulfill the needs of new operating system APIs must fulfill a number of requirements. They are:

1. Discoverable and accessible through self-service
2. Productized and designed for ease of consumption
3. Easily managed for security and performance

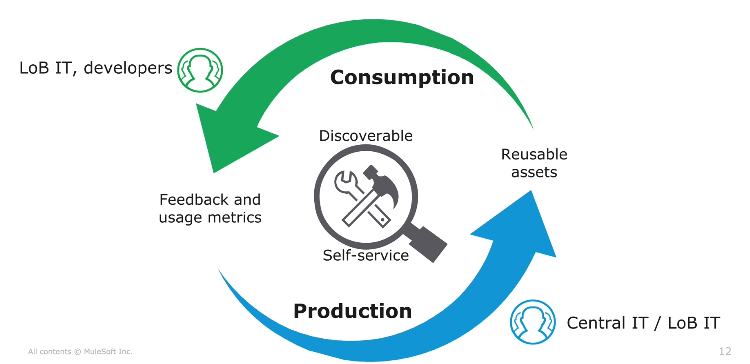
* API LED CONNECTIVITY APPROACH :

Modern APIs are core building blocks of new operating systems. Building applications using this approach is known as API Led Connectivity approach. Instead of building 1 application that encapsulates all functionalities and connections, this approach uses groups of services, each having its own API that can be controlled and monitored. This approach is very useful while creating mobile apps as we don’t have to start from the scratch like in earlier scenario. We can use the existing APIs in our mobile application by calling them and retrieve the existing data from the core databases using the System APIs.

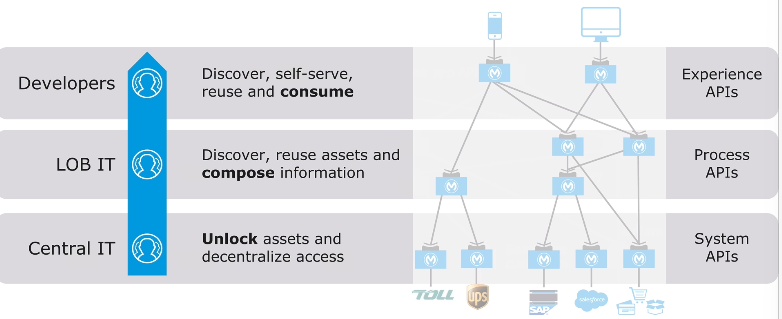


This approach contains 3 stages. They are:

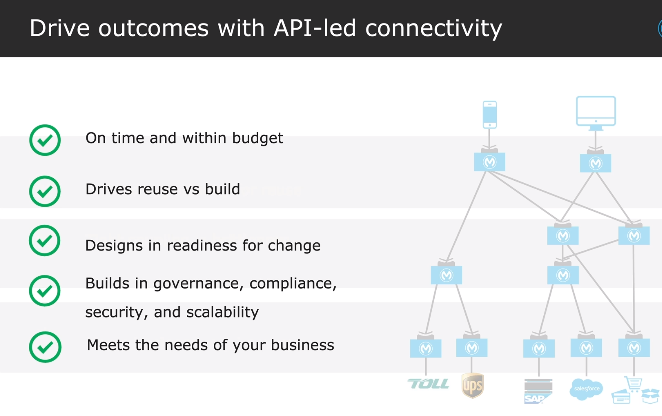
1. Experience APIs
2. Process APIs
3. System APIs

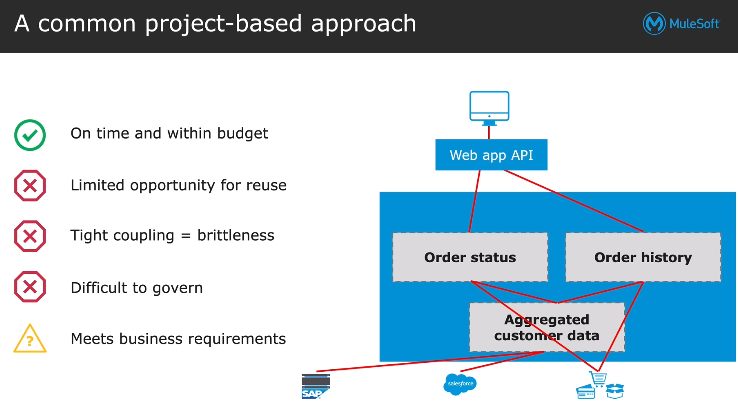


By using this approach the members of Central IT (or) LOB produces reusable assets that can be used by the entire organization and are also made available to the rest of the organizations. The end result of this approach is to enable and empower the entire organization.



Central IT unlocks the data in the core system by producing System APIs whose access can be monitored and governed. LOB IT discovers and consumes data that are made accessible via APIs System APIs and then creates additional APIs. These are Process APIs that compose data, transforming, enriching, aggregating and orchestrating as needed. These Process APIs are now reusable by others as well. The Developers that are going to build the build the new apps can discover, self-serve, reuse and consume the existing APIs and the build the Experience layer APIs.





* C4E :

Center for enablement (C4E) can be used to adopt the API Led Connectivity. C4E is a cross functional team and shrinks that all the applications are created from the existing assets as far as possible and produces new assets as needed for functionality (or) connections that don’t exist. They are responsible for promoting the consumption of assets across the organization. C4E ensures that assets are productized and published, consumable, consumed broadly, and fully leveraged. The success of C4E is based on the asset consumption.

* APPLICATION NETWORK :

The other benefit of using API in the organization is that an application network is created. This is the network of applications, data and devices that are connected with APIs. The power of this network is, every time a node (or) API is added to the network, the number of possible connections increases not by 1 but multiplicatively. This means, adding a new node (or) API increases the value of the network. If a project is build using API Led Connectivity, network nodes are created. These nodes (or) APIs acts as endpoints for the other applications to connect to. Each time a new project is built, new nodes are added to the network. If there are 4 nodes then there are 8 connections and if a 5th node is added then there are 10 connections. The benefit of API Led Connectivity to build an application network is that it takes only 1 project at a time.

The application network emerges from the bottom up and APIs are added to the network. The key to the success of the application network is that all the APIs are easily discoverable so that they can be consumed in a self- service fashion but all the members in the organization. The application network is easily secured and monitored by the Central IT. It provides 4 control visibility of who can control what and what is being consumed when. This enables the entire organization to rapidly innovate and respond to technology and market pressures, providing the infrastructures for the speed and agility needed to transform our business and stay comparative in today’s market. It is recomposable – it bends, not breaks. That is, it’s built for change.

* DECONSTRUCTING APIs:

APIs can be easily managed for security, scalability and performance. API stands for Application Programming Interface. There are introduced since the beginning of programming languages in the 50s (or) 60s and the rules of modern APIs are same as that of the legacy APIs. It provides the information for how to communicate with a software component. It requires operations (what to call), inputs (what to send with a call), outputs (what we get back from a call), and underlying data types. It defines the functionalities independent of the implementations. We can change what is going on behind the scenes without changing how people call it.

API could be referring to a number of things. Modern API contains all these three things. They are:

1. **API interface definition file (API Specification) :** It define what we can call, what we send it and what we get back.
2. **A web service :** The actual API implementation typically made available as a web service to which we can make calls to (or) the interfaces of that API implementation.
3. **An API proxy :** An application that controls access to web service, restricting access and usage through the use of an API gateway.

* WEB SERVICE:

A web service is a method of communication that allows two software systems to exchange data over the internet. Systems interact with the web services in a manner prescribed by some defined rules of communication like how one system can request data from other, what parameters are required, the structure of return type, etc..

* **Parts Of A Web Service :**

1. **The web service API :**

* Describes how we interact with web service : It is a comprehensive description of how a client interacts with the application.
* The API should preferably defined in a file using any type of text representation.
* It could be any type of txt in any type of file but ideally a standard API description language such as RAML (or) OAS.

1. **The web service interface implementing the API :**

* This code provides the underlying structure to the application so that the described API is implemented properly.
* This may be combined with the actual implementation code.

1. **The web service implementation itself :** This contains the actual code and application. In theory, this code is different from the code implementing the API. But in practice, the 2 code basis are sometimes combined.

* **Types Of Web Services :**

1. **SOAP web services :**

* Traditional and more complex and requires we to know 2 protocols. One is SOAP, which is used to encode and decode the responses and the other is either HTTP (or) another protocol which is used to communicate those requests and responses with the client and server.
* The communication rules are defined in an XML based WSLD (web Services Description Language) file.

1. **REST web services :**

* They are simpler type of web services and uses the existing HTTP communication protocol.

SOAP WEB SERVICES :

SOAP stands for Simple Object Access Protocol. SOAP is a way for a program running in one operating system to communicate with a program running in either the same (or) a different operating system, using HTTP (or any other transport protocol) and XML. SOAP message consists of three parts. They are :

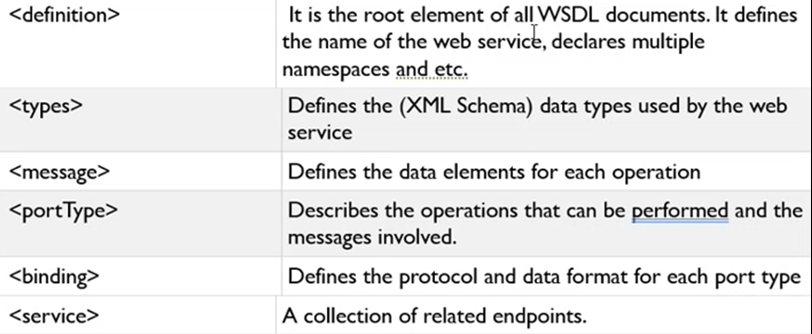
1. **SOAP Envelope :** Envelope is used to define the start and end of the SOAP message. It contains the details of the SOAP message. It is an important element of the XML documents. The SOAP Envelope construct defines an overall framework for expressing what is in a message and who should deal with it. The Envelope element is always the root element of a SOAP message.
2. **SOAP Header :** It is an optional element in which the header contains the credentials information such as authorization, authentication, etc. is used during the processing of a SOAP message. The Header element is container for control information. It may contain any number of elements from any namespace and should contain information that influences payload processing.
3. **SOAP Body :** It is an important element of the SOAP message that contains request and response information in the XML format. It defines the actual content of the message to be sent between the client and the web server. The Body element represents the message payload.
4. **SOAP Fault :** The SOAP Fault element is an optional element used to display an error message encountered during the transmission of a SOAP message. It holds the status of SOAP messages and errors. The following are the sub-elements of the SOAP Fault element.

* **WSLD Document :**

|  |  |
| --- | --- |
| **Sub Fault Element** | **Description** |
| **<faultcode>** | It is used to identify the fault code in the SOAP message. |
| **<faultstring>** | It is used to provide the human-readable description of the error. |
| **<faultactor>** | It is an optional element in the SOAP Fault that indicate the fault occurred during processing if the message. |
| **<detail>** | It is used to hold the application-specific status error of the Body element. |

WSDL stands for Web Services Description Language. WSDL is used to describe SOAP web services. WSDL is written in XSD format.

**WSLD Components :**



REST WEB SERVICES :

REST stands for Representational State Transfer. It is an architectural style that defines how different systems communicate with one another over HTTP Protocol. Some of the characteristics of REST are servers are stateless, that is, the server does not remember any client state from previous requests. The clients can cache previous responses to avoid repeated network calls. Other systems interact with the web service using the HTTP protocol. The HTTP request method indicates which operation should be performed on the object identified by the URL. Data and resources are represented using URIs and resources are accessed (or) changed using a fixed set of operations. A Restful web service can turn data in a number of media types like JSON, XML, Java etc…

Example :



1st – retrieve a collection of companies

2nd – query parameter to get collection of companies in France

3rd – URI parameter to get the collection with unique id 3

4th – it is used to create a new company in the collection of companies

5th – deletes the company with id 3

6th – it is used to replace the company with id 3 (or) create one if it is not present

* **APIs contain the following :**

1. **API Documentation :** It should include the list of all possible resources, how to get access to the API, and more.
2. **API Portals :** Accelerate on boarding by providing developers a centralized place for discovering all the tools they need to successfully use the API, which could include :

* Documentation, tutorials, code snippets, and examples.
* Away to register applications to get access to the API
* Away to provide feedback and make requests
* Away to test the API by making calls to it

1. **Discover APIs in API directories and market places :** For example, APIs.guru, which has thousands of APIs.

* **Calling restful web services :**

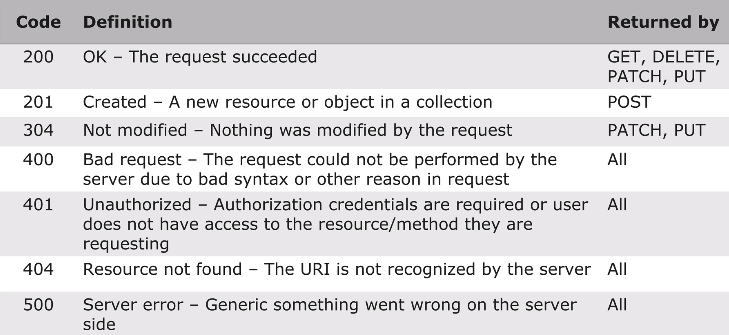
To call web services, we need to write code (or) have a tool to make the HTTP request. We need to be able to specify the HTTP method, Request headers and request body. To call web services, we can use a client tool such as an API portal with API console, Advanced Rest Client, Postman (or) cURL, a command line tool that can be scripted. We can make calls to both secured and unsecured APIs.

1. **Unsecured API :** The API may be public and requires no credentials to call them (authentication not necessary).
2. **Secured API :** The API may be secured and requires some sort of authentication to access them. We have to provide either a token (or) credentials such as username and password etc.. Often a proxy is created to govern access to an API. We will call and then later create an API secured by credentials. You can also secure an API with other authentication protocols like SAML, JWT, OAuth etc..

* **Getting responses from web service calls :**

Restful web services return an HTTP status code with response. The status code provides client feedback for the outcome of the operation (succeeded, failed, updated). A good API should return status codes that align with the HTTP spec.

HTTP STATUS CODES :



HTTP METHODS :

1. **GET** - This method retrieves data by using either URL (or) query (or) URI parameters. URI stands for Unique Resource Identifiers and they are used for identification. Query parameters are in the key value format and are used for filtering the data. The response is generated based on the request. This method retrieves the current state of a resource in some representation (JSON (or) XML). The success response for GET method is 200 OK.
2. **POST -** This method creates a new resource. Here the body needed. The success response for this method is 201.
3. PUT - This method is used to update a resource. It checks if the resource is present (or) not. If the resource is present then it updates else creates a new resource and updates. Here, body is used to pass the data to the resource. The success response for this method is 200.
4. **PATCH** - This method directly updates the resource even if it is there (or) not. It is used to partially update a resource. Here, body is used to pass the data. The success response for this method is 204.
5. **DELETE** - This method delete a particular resource. Here, body is not needed and only query (or) URI are passed like GET method. If a DELETE method is successfully applied, there are several response status codes possible:

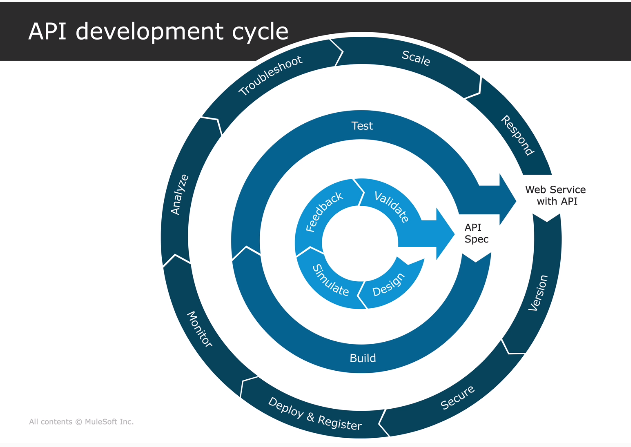
* A [202](https://www.devdoc.net/web/developer.mozilla.org/en-US/docs/Web/HTTP/Status/202.html) (Accepted) status code if the action will likely succeed but has not yet been enacted.
* A [204](https://www.devdoc.net/web/developer.mozilla.org/en-US/docs/Web/HTTP/Status/204.html) (No Content) status code if the action has been enacted and no further information is to be supplied.
* A [200](https://www.devdoc.net/web/developer.mozilla.org/en-US/docs/Web/HTTP/Status/200.html) (OK) status code if the action has been enacted and the response message includes a representation describing the status.

DESIGNING FOR API SUCCESS :

The whole key to the new operating model is making assets that are easy to find, easy to use and will form the application network. Producers need to make assets and APIs that developers can find and want to use and share with others. We should always keep in mind that, we should design the API based on business use cases, which will fulfill but not based on the model the backend services (or) applications they expose. We should focus on performance of client applications and user experience. It is important to create APIs taking design first approach. This means, first design the API correctly before investing in building it and once it is complete, we should define it iteratively & get feedback from the developers on its usability and functionality along the way before moving onto the implementation. Building implementation of an API is time consuming and expensive to undo.

API DEVELOPMENT CYCLE :

API Life Cycle has 3 things. They are : Design, Implementation and Management.



The first phase deals with the API Specifications. We start with the design phase where we will receive the requirements and then design the API specification. Next we go to the simulate stage where we create the prototype for implementation. The prototype is then made accessible to the users who can provide feedback which helps the designer determine if they have considered all of the consumers needs. After the feedback stage it is validated. The output of this stage is the API Specification some times referred to as contract.

The second phase deals with the API Implementation where the specification is used to build and test the API functionality. At the completion of this stage, the API is ready to be deployed as a web service.

The final phase deals with poing and managing the web services. It starts with establishing the API version, creating policies for security, deploy and registering the web service with the API, monitoring the operations, analyzing and reporting on the usage, troubleshooting ans scaling to meet the resource demands and responding to the changing needs of the organization by repeating the life cycle.

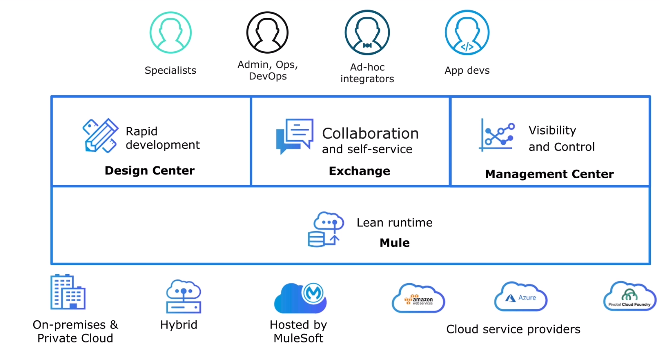
THREE LAYER CONNECTIVITY ARCHITECTURE :

There are 3 layers in this architecture. They are :

1. **Experience Layer API** : frontend, inside unknown but we get response, frontend req to exp layer, exp to process, process to system, process output exp input
2. **Process Layer API** : multiple info takes, converts according to user requirements, system output process input
3. **System Layer API** : backend, not known to user, all implementation here only, connects to database and retrieve data and send that data to process layer then exp layer then to user, exp layer output system input, if there is only 1 system API then experience layer is directly connected without process layer

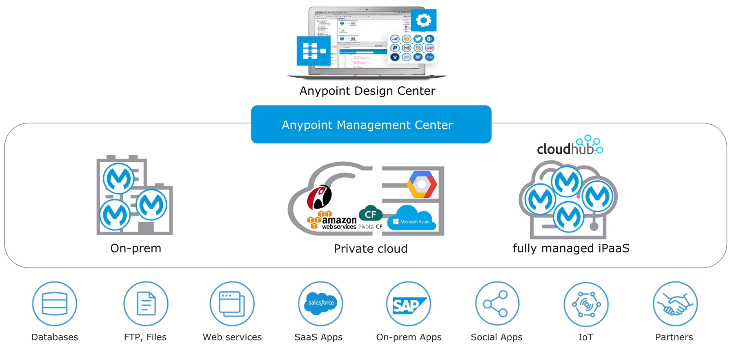
ANYPOINT PLATFORM :

Anypoint Platform has all the components needed for an entire API life cycle (to design, build, deploy and manage). It uniquely enables the building of an application network. Anypoint Platform is a unified, hybrid integration platform that creates a seamless application network of apps, data, and devices with API Led Connectivity. The modern APIs are productized for the ease of consumption using the design center. They are discoverable and accessible to self service using the exchange, and are easily managed for security, scalability and performance using management center. Anypoint Platform is the most advanced enterprise platform for designing, developing and managing APIs and integrations. It can be deployed anywhere through reliably addressable small scale use cases.

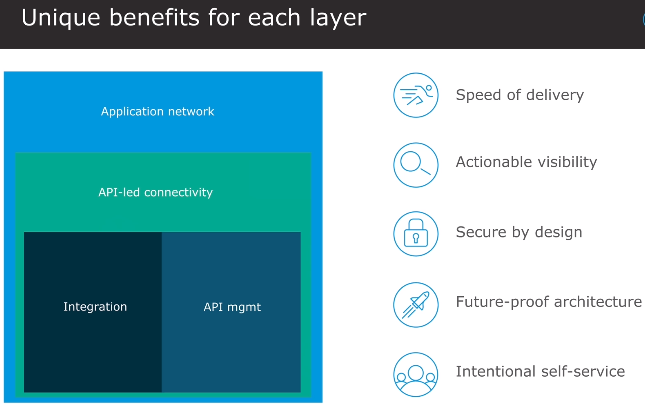


A single platform supports the need of every row such as specialists, admin, operations, DevOps, Ad-hoc integrators and application developers. Exchange is used to collaborate, discover assets and get stakeholder feedback. Management center is used to manage different aspects of the applications. Supporting all these, the mule runtime is the application server for mulesoft application. ﻿Spec driven development is a development process where your application is built in two distinct phases. They are :

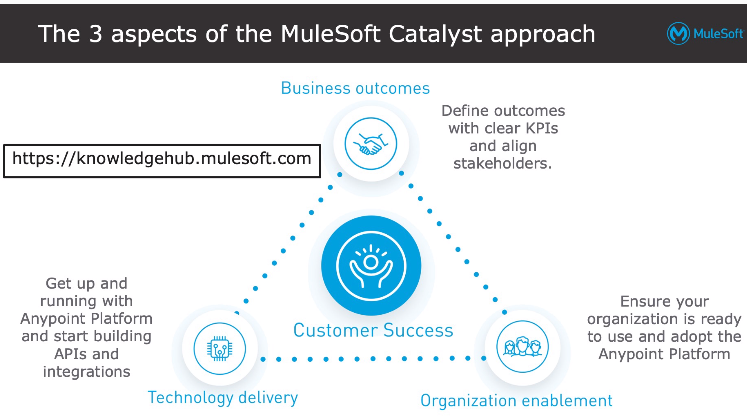
1. The creation of a spec (the design phase)
2. Development of code to match the spec (the development phase)

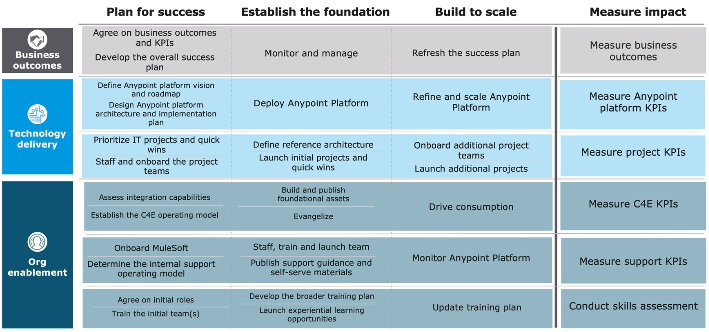


This platform allows us to design once and deploy anywhere including customer hosted environments such as on-premise, private cloud and cloudhub, which is a fully managed iPaaS solution. Regardless of where the application is hosted it contains the same connectivity options such as databases, web services, social apps etc.. The application network built using API Led Connectivity uses integration and API management.

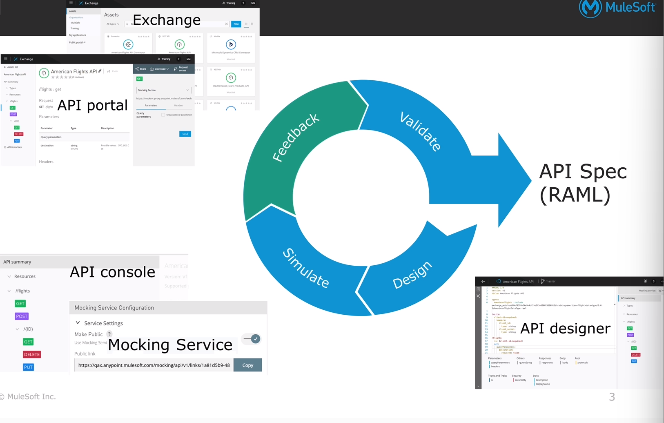


Mulesofts approach to customer success is Mulesoft Catalyst approach, formally known as OBD (Outcome Based Delivery). It is centered around 3 core pillars. First is to identify the business outcomes using KPIs and align them to the key stakeholders. Second, delivering technology using the right approach for each project (or) use case and at the same time building a platform to drive longer term benefits. Lastly, enabling the organization to use the anypoint platform to achieve those business outcomes.





The tools for API specification phase are :



1. **API Designer :** The design and simulate stage are accomplished using API designer. The RAML is used to create the API specification. It is used to describe the contract that provides all of the details about the operations, inputs and outputs.
2. **API console and mocking service :** The designer has a console with an embedded mocking server. The console displays a visual representation of the API and the mocking service is used to test the design using example data. They are used to interact with and simulate the API. The console is also used to display errors and warnings.
3. **API portal and Exchange :** The specification is published to the Exchange which creates an API portal making it available to the other stakeholders, which allows them to provide feedback.
4. **API notebook :** It can be added to the portal to combine documentation with java script code to demonstrate how the API can satisfy a variety of use cases. The output of this phase is the validated API specification created with RAML.

The tools for API implementation phase are :

1. **Anypoint Studio :** It is a user friendly IDE used to import the API specification and build Mulesoft application.
2. **Composer :** It is a web based tool that enables citizen integrators to create integration solutions with clicks not code. It enables an integrator to easily build flows and automate integration tasks.
3. **MUnit :** The embedded MUnit framework can be used to automate unit testing.

The tools for API management phase are :

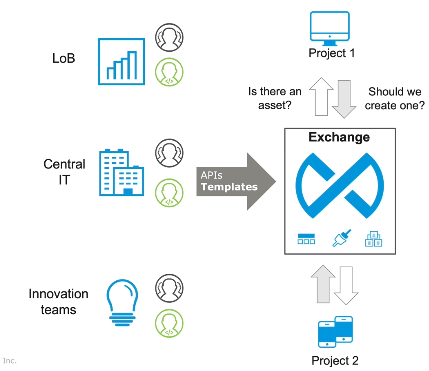
1. **API analytics :** It helps to analyze and report on how the applications are being used.
2. **API manager :** It is used to create a proxy application which is used to establish the initial version and update to new versions of the API specification and is also used to govern and secure the APIs by creating policies that can control the access and filter requests.
3. **Runtime manger :** It is used to deploy and register the web service and monitor its performance.
4. **Anypoint monitoring :** It provides additional monitoring which provides performance visibility into your application network through aggregated matrix, data visualization tools, alerts for issues and a log aggregation system.
5. **Visualizer :** Troubleshooting and scaling is accomplished using API manager, runtime manager and visualizer. It provides a real time view of our application network and enables identification of root cause issues quickly.

These steps are repeated to update (or) create new versions of the API.

EXCHANGE :

It is a library of assets. It enables the production- consumption model to be successful, it does this by providing all the users to discover and self serve these APIs as other assets and can consume them using Anypoint Studio. Here, discovery is supported by strong search capabilities which can act in two nodes. The newer search experience offers features such as searching by using API specification properties and the ability to save searches. MuleSoft provided public assets are available in all accounts to all users, we can work with MuleSoft to get APIs and connectors certified and added. Private contents are only available to people in your organization. Assets added by anyone in your org are added to your private Exchange. Your organization should populate it to contain everything you need to build your integration projects including APIs, connectors, diagrams, videos, links, etc...

When a REST API is added to Exchange, an API portal is automatically created for it. An API portal has auto generated API documentation, an API console for consuming and testing APIs, an automatically generated API endpoint that uses a mocking service to allow the API to be tested without having to implement it. API portals can be shared with both internal and external users. When a RAML 1.0 API specification is added to Exchange, a connector is automatically created for it. The connector can be used in Mule applications to make calls to that API. REST Connect is the name of the technology that performs this conversion.



The Central IT publishes the APIs and templates to the Exchange and these can be used by the other projects and the projects can also publish their assets to the Exchange.

DESIGN CENTER :

It includes 2 applications. They are :

1. **API Designer :** It is made for developers to design, document and mock APIs.
2. **Anypoint Studio** **:** It is a development tool used for implementing APIs and building integration applications.

Mule applications can be created using Anypoint Studio. Studio creates XML code visually by adding components and processors to flows. The XML can also be manually edited (or) created. We can use connectors and other assets from Exchange. Other tools can be used to write code (primarily XML) to create applications. Under the hood, Mule applications are Java applications using Spring.

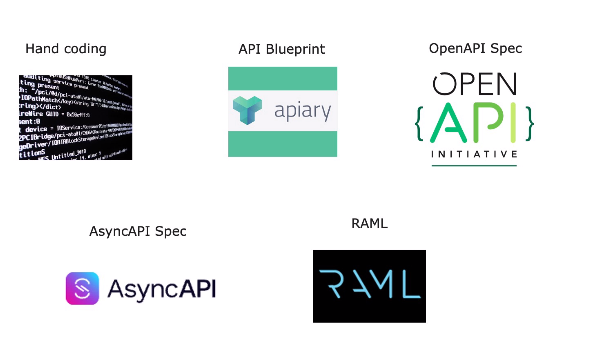
MULE RUNTIME ENGINE :

It is a lightweight integration and automation platform that allows developers to connect applications together quickly and easily, enabling them to exchange data. It acts as a transit system for carrying data between apps (the Mule). It can connect all systems including web services, JMS, JDBC, HTTP, & more. It decouples point-to-point integrations by having all (non-Mule) apps talk to a Mule runtime instead of directly to each other. It is used to enforce policies for API governance. It can be deployed anywhere like On-premises & Private Cloud, Hybrid and Hosted by MuleSoft (CloudHub), can integrate and orchestrate events in real time (or) in batch, and has universal connectivity. ﻿

Mule applications are designed to run on Mule. Mule applications are created by integration developers to tie together various subsystems. These applications are deployed to a Mule Runtime which enables them to consume inbound data in a predefined Mule message format. The applications transform and route Mule messages in paths called flows and in stages called components (or) processors. Mule finally delivers the transformed data to a recipient (or) destination. Mule applications can be deployed to anywhere a Mule Runtime is hosted. Mule runtimes can be MuleSoft hosted in the cloud (CloudHub), private-hosted in the cloud, on-premise, (or) a hybrid.

WAYS TO DESIGN API :

There are several ways to design the API. They are :

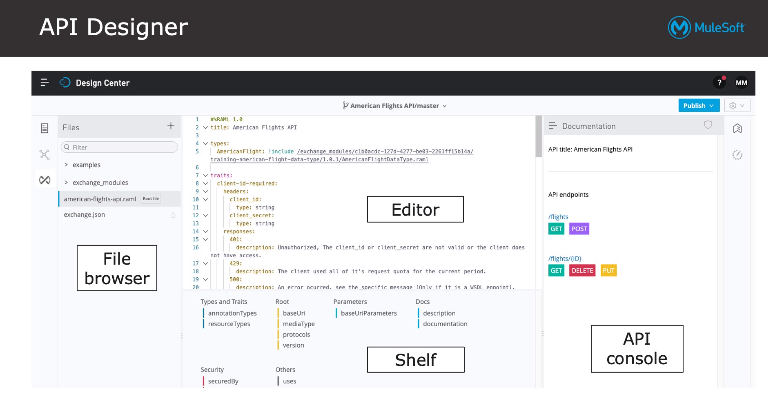


1. **Hand Coding :** It is the initial approach used by the developers.
2. **API Blueprint :** Apiary tool is used to create API blueprint. It looks like documentation, uses markdown and structure of API itself gets lost.
3. **Open API Spec :** Swagger tool is used to create an open API specifications. It is JSON based, descriptive, generate OAS from code, great for documentation, design center supports API specification development using OAS.
4. **Async API Spec :** Event driven API Specification can be developed using Async API. It is a language that describes the messaging interfaces, facilitates creation of event driven architectures, open source, industry standard, technology agnostic, supported by design center and Anypoint Exchange.
5. **RAML :** It is an open source, API design in an asynchronous human centric language and supports structured files in inheritance.

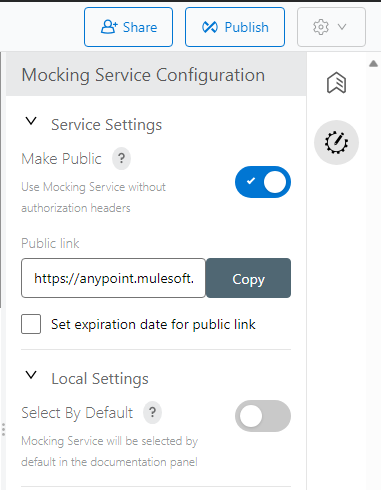
RAML :

﻿RAML is a simple, structured, and succinct way of describing RESTful APIs. It is a non-proprietary, vendor-neutral open spec. It is developed to help out the current API ecosystem by encouraging reuse, enabling discovery and pattern-sharing, and aims for merit-based emergence of best practices. RAML files can be used to auto-generate documentation, mocked endpoints, interfaces for API implementations. ﻿

RAML syntax is based on broadly-used standards such as YAML and JSON. It uses a human-readable data serialization format where data structure hierarchy is specified by indentation, not additional markup characters. The indentation shows a parent child relationship between resources and methods. RAML syntax consists of nodes and facets. Resources are the objects identified by the web service URL that you want to act upon using the HTTP method used for the request. Resources are nodes that begin with a slash. Any methods and parameters nested under a resource belong to and act upon that resource. Nested resources are used to select members from the collection and for a subset of a resource to narrow it. Facets are special configurations that can be applied to nodes. URI parameter are placeholders enclosed in {} braces.



SIMULATING THE API :



﻿We can mock an API to test it before it is implemented. This is useful to get early feedback from developers. We can use the API console and the mocking service to run a live simulation which returns sample API responses defined in the API definition. Mocking is available to outside stakeholders through shareable links, an expiration date can be set if needed. The API console is available in API Designer, so the API designer can test API portals in Exchange, so users/developers can test it. The public link can be copied and endpoint is added, this link works everywhere like edge, chrome, postman etc..

RAML REQUEST AND RESPONSES :

﻿Responses are facets which must be a map of one (or) more HTTP status codes. For each response, specify possible return data types along with descriptions and examples. ﻿For a request, similarly specify the possible request data types along with descriptions and examples. ﻿There are two optional facets you can use to specify example data: example and examples. We use example to represent a single instance of the data and examples to represent multiple instances of the data as a map of key-value pairs.

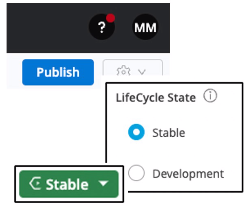
|  |  |
| --- | --- |
| EXAMPLE :  type: User  example:  name: Bob  lastname: Marley | EXAMPLES :  type: User  examples:  person1:  name: Paul  lastname: McCartney  person2:  name: Lady  lastname: Gaga |

﻿Instead of including all code in one RAML file, you can modularize it and compose it of reusable fragments like data types, examples, traits, resource types, overlays, extensions, security schemes, documentation, annotations, and libraries. Fragments can be stored in different files and folders within a project (or) in a separate API fragment project in Design Center (or) in a separate RAML fragment in Exchange. To build a successful API, you should define it iteratively, get feedback from developers on usability and functionality along the way.

For this, we need to provide ways for developers to discover and play with the API. Anypoint Platform makes this easy with API portals in Exchange. Publish in private Exchange for internal developers, to share an API within an organization and in a public portal for external developers.

PUBLISHING API TO EXCHANGE :

We can publish RAML API specifications and other assets to Exchange from API Designer, not from Exchange itself. When publishing assets, you specify their state in the developmental lifecycle like :

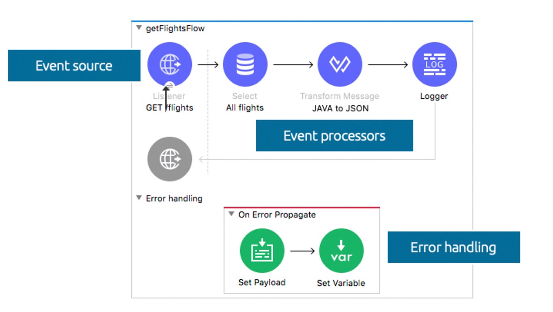


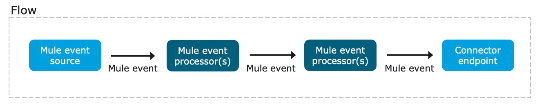
1. **Development:** still in the iterative design process
2. **Stable:** ready for consumption, for APIs, required to enable Studio import (or) API Manager governance

API portals are automatically created for REST APIs added to Exchange. An API console can be used for consuming and testing APIs. An automatically generated API endpoint, that uses a mocking service to allow the API to be tested without having to implement it. API portals can be shared with both internal and external users. Anypoint Exchange provides a uniform and simplified way to navigate the asset portals for all types of assets like APIs, Templates, Connectors and Examples. Navigation is based on minor version. Asset properties are associated with each minor version of the asset like Tags, Categories, Markdown page, Custom fields etc..

MULE APPLICATION AND FLOWS :

Mule applications receive events, process them, and route them to other endpoints. Mule applications accept and process a Mule event through a series of Mule event processors plugged together in a flow. An application can consist of a single flow, multiple flows (or) multiple flows connected together. A Mule application is a sequence of Mule event processors. These processes are again organized under another processor called flow. A flow is the only thing that can execute a mule application. The flow has 3 main areas. They are : source, process area and error handling. The source area controls how a flow is triggered. Once the source is triggered, the process area determines what it does. A mule application is a set of XML files which can have multiple flows. Mule event processors transform, filter, enrich, and process the event and its message.



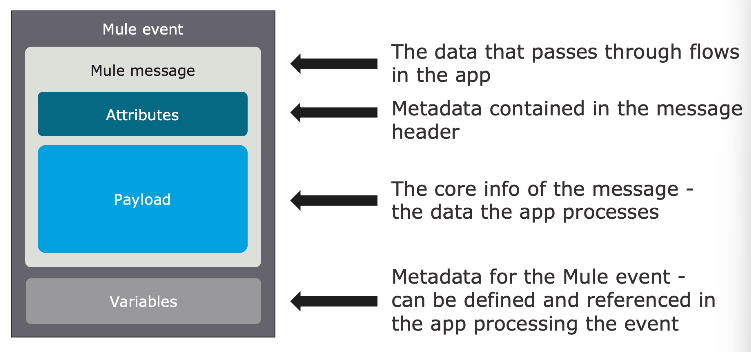


A Mule event source that initiates the execution of the flow can be triggered by an event such as

1. A consumer request from a mobile device
2. A change to data in a database
3. The creation of a new customer ID in a SaaS application

MULE 4 EVENT STRUCTURE :

When a new application is triggered, a data structure is created and passed through flows in the application. The data structure is known as mule event. The mule event has 2 main elements. They are mule message and variables. The mule message again contains 2 components. They are attributes and payloads. The attributes are considered to be the metadata describing details about the payload. The payload is the actual data to be processed. If a post HTTP request is sent to the application, the payload would contain the data to create a new resource and the attributes would contain the parameters and headers. Variables are created and managed by the processors within the flow.

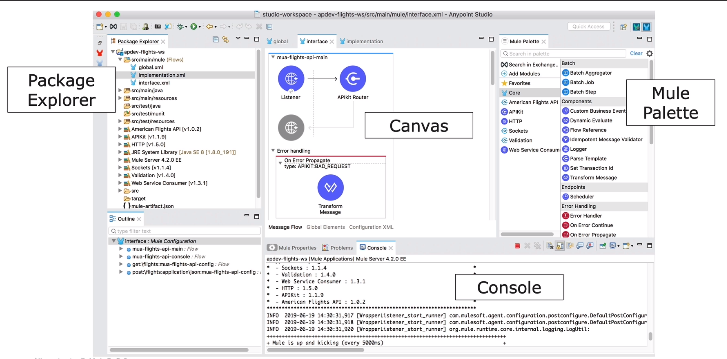


ANYPOINT STUDIO :

It is based on Eclipse, a common Java integrated development environment.

* **Features included here are :**

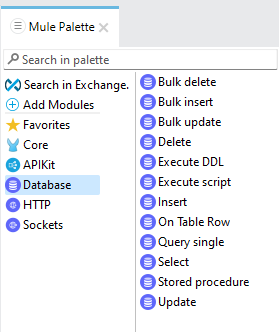
1. Two-way editing between graphical and XML views
2. Pre-built tooling to connect to APIs (REST, SOAP), protocols (HTTP, FTP, SMTP, more), and popular services (Salesforce, Workday)
3. A data transformation framework and language
4. An embedded Mule runtime to test applications without leaving it
5. Visual debugging
6. Round-trip editing of API specifications with Anypoint Platform
7. One-click deployment of applications to CloudHub
8. Templates for common integration patterns
9. Integration with Maven for continuous build processes



1. ﻿**Package Explorer :** It contains the list of files and folders. The window below package explore displays the flow.
2. **Canvas :** It is the place where the mule applications are created. It contains 3 parts. They are : message flow, global elements and configuration XML. The mule project is run or deployed after saving it. The components can be commented using toggle comment option.
3. **Console :** Anypoint Studio comes with an embedded Mule runtime to test applications without leaving it. The console outputs application logs and information. It is also used to display the error in the application.
4. **Mule palette :** Are separated into categories in the Core section of the Mule Palette. By default, projects include HTTP and Sockets modules. We can also add additional modules for more functionalities just by drag and drop (or) we can also download them outside the studio. Mule applications are built using connectors and processors that are contained in the palette. The modules are used to interact with HTTP, TCP and UDP protocols.
5. **MUnit :** We can automate testing of Mule applications using MUnit. MUnit is a Mule app testing framework for building automated tests. MUnit is fully integrated with Anypoint Studio and we can create, design, and run MUnit tests and suites of tests just like we do Mule applications.

DATABASE CONNECTOR :

The database connector can connect to almost any JDBC relational database, any database engine for which you have a driver. To use the database, we have to add the Database module to your project and then add a database operation to a flow and configure the connection to the database. For most operations, a lot of the configuration is encapsulated in a separate global element, a reusable configuration that can be used by many operations. It defines a connection to a network resource. This is a connector configuration though it is sometimes referred to simply as the connector.



The database has many options such as :

1. **INSERT :** This component is used to insert 1 row at a time. We cannot insert multiple rows using this component. Here we need to provide the parametrs with :(colon) infront of them. insert as in mysql except values.

Example : "cid":payload.name\_as\_in\_table

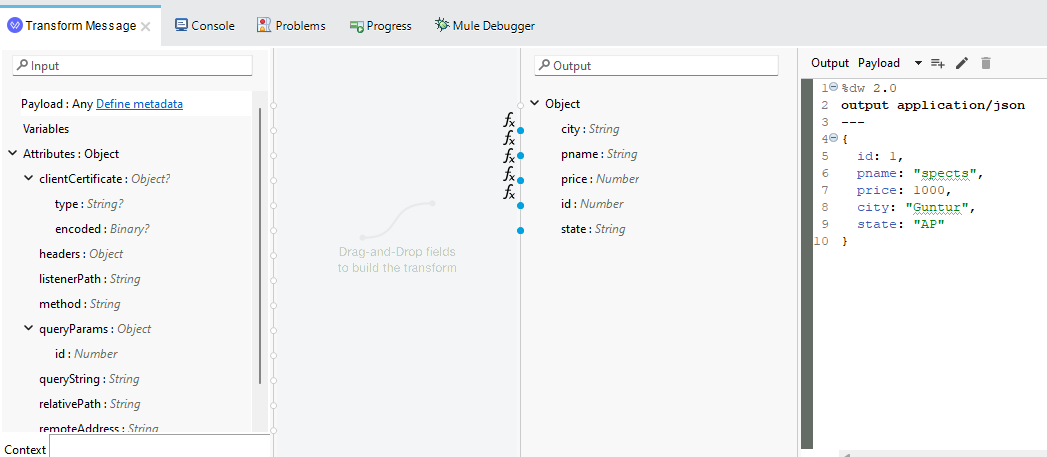
1. **BULK INSERT :** This component is used to insert multiple rows at a time. Here you don't have to provide parameters also.
2. **DELETE :** This component is used to delete 1 row at a time. We cannot delete multiple rows using this component.
3. **BULK DELETE :** This component is used to delete multiple rows at a time.
4. **SELECT :** This component is used to retrieve the data from the database. It's working is same as that of MYSQL.
5. **STORED PROCEDURES :** This component is used to perform multiple operations at the same time just by calling the function.
6. **UPDATE :** This component is used to update only 1 column at a time.
7. **BULK UPDATE :** This component is used to update multiple columns at a time.

TRANSFORM MESSAGE :

DataWeave 2.0 is the expression language for Mule to access, query, and transform Mule 4 event data. DataWeave is fully integrated with Studio

1. Graphical interface with payload, aware development
2. In Studio, the Transform Message component is used for transformations

DataWeave Playground is an interactive browser environment for mocking. We use the transform message to convert the output to Java (or) JSON (or) Text (or) XML based on our choice. We can define the input and output in the transform message and we can use visual mapping to get the script automatically. A RESTful interface for an application will have listeners for each API. We can create the interface manually or have it generated from the API definition. Visual data mapping only takes mapping, it does not provide other functions like filter. Visual data mapping is not usually recommended method.



APIKIT :

﻿APIKit is an open-source toolkit that includes an Anypoint Studio plugin. The Anypoint Studio APIkit plugin can generate an interface automatically from a RAML API definition for new or existing projects and can also work with OAS. It generates a main routing flow and flows for each of the API resource / method pairs. You add processors to the resource flows to hook up to your backend logic. APIKit router routes them based on the methods and generates private flow for the methods.

FLOW REFERENCE :

﻿Flows can be broken into multiple flows

1. Makes the graphical view more intuitive and the XML code easier to read
2. Promotes code reuse
3. Easier to test with MUnit

All flows are identified by name and can be called via Flow Reference components in other flows. Studio can list all flow references to a flow or subflow and also provides navigation.

API SYNC :

﻿API Sync feature of Anypoint Studio enables us to pull specifications from Design Center into Studio. We can also initiate the creation of API specifications from scratch in Studio

1. Edit the specification offline in Anypoint Studio
2. Push the updates back to Design Center
3. Publish the new API asset version to Exchange

This lets you develop Mule applications while following API lifecycle development practices from within Anypoint Studio. If an API specification changes in Exchange, the generated API interface in Anypoint Studio can be updated. Flows that have already been modified are not overwritten. The API Design perspective includes a Git Staging tab. Under the hood, a Git version control system is used to pull, push, and merge branches made to API specifications. If someone modifies the version in Design Center while you are modifying the same version locally, a conflict is triggered in Git You must tell Git how to apply your changes over a modified version.

DEPLOYING THE APPLICATION :

During development, applications are deployed to an embedded Mule runtime in Anypoint Studio. For everything else (testing, Q&A, and production), applications can be deployed to

1. CloudHub & CloudHub 2.0 :

* Platform as a Service (PaaS) component of Anypoint Platform
* MuleSoft-hosted Mule runtimes on AWS
* Uses virtual machines on AWS platform
* A fully-managed, multi-tenanted, globally available, secure and highly available cloud platform for integrations and APIs

1. Customer-hosted Mule runtimes :

* On bare metal or cloud service providers: AWS, Azure, and Pivotal Cloud Foundry

1. Anypoint Runtime Fabric :

* Customer-hosted container service of the runtime plane



* **CloudHub Benefits :**

1. No hardware to maintain
2. Continuous software updates
3. Provided infrastructure for DNS and load-balancing
4. Built-in elastic scalability for increasing cloud capacity during periods of high demand
5. Globally available with data centers around the world
6. Highly available with 99.99% uptime SLAs (service level agreements)
7. Highly secure - PCI, HiTrust, and SSAE-16 certified

* **Customer Hosted Mule Runtimes :**

1. Easy to install
2. Requires minimal resources
3. Can run multiple applications
4. Uses a Java Service Wrapper that controls the JVM from the operating system and starts Mule
5. Can be managed by Runtime Manager in MuleSoft-hosted Anypoint Platform and Runtime Manager in customer-hosted Anypoint Platform
6. Anypoint Platform Private Cloud Edition

* **Anypoint Runtime Fabric :**

1. A deployment model of the runtime plane

* Container service that automates orchestration of apps and API gateways
* Anypoint Platform control plane is still hosted by MuleSoft

1. Runs on customer-hosted infrastructure - AWS, Azure, virtual machines (VMs) or bare-metal servers
2. Main characteristics and capabilities

* Security and reliability through isolation between applications
* Re-deployments with zero downtime
* Automated application failover through horizontal scaling
* Run multiple versions of the Mule runtime
* MuleSoft-supported containerized runtime images
* **Viewing Deployed Applications with Vizualizer :**

Visualizer provides a real-time view into your application architecture in a context that best suits your role. Diagram data is secure and automatically & dynamically updated. Organizes APIs and applications into relational diagrams

1. Promotes best practices and layer-based architectural consistency
2. Pinpoints issues rapidly through root cause analysis
3. Enables visibility into the policy compliance of APIs

Anypoint Monitoring provides visibility into integrations across your app network. Its monitoring tools are designed to reduce the time to identify and resolve issues by providing ways to aggregate and map metrics across dependent systems in real-time and configure dashboards and alerts to reduce issue identification time, store and search log data at scale.

DEPLOYING APPLICATIONS TO CLOUDHUB :

We can deploy from Anypoint Studio or from Anypoint Platform using Runtime Manager. You must set worker size and number. A worker is a dedicated instance of Mule that runs an app. Each worker runs in a separate VM from every other application and is deployed and monitored independently. They run in a specific worker cloud in a region of the world. Workers can have a different memory capacity and processing power

1. Applications can be scaled vertically by changing the worker size
2. Applications can be scaled horizontally by adding multiple workers

There are workers in different environments like Design, Sandbox, Production, etc.. Apps can be promoted from one environment to another.

SCALING :

There are 2 types of scaling. They are :

1. **Horizontal Scaling :** This requires less CPU allocations and memory is also free. Load balancer is an essential thing here. This is done for availability because you do not want a single point of failure.
2. **Vertical Scaling :** This requires more CPU allocations which in turn requires memory.

PROXY :

An API proxy is an application that controls access to a web service, restricting access and usage through the use of an API gateway. The API Gateway is a runtime designed and optimized to host an API or to open a connection to an API deployed to another runtime. It is included as part of the Mule runtime. It separates licenses required and separates orchestration from implementation concerns. The API Gateway is the point of control. It determines which traffic is authorized to pass through the API to backend services, meters the traffic flowing through logs all transactions, collecting and tracking analytics data. It applies runtime policies to enforce governance like rate limiting, throttling, and caching.

* **Using API Manager To Manage Access To APIs :**

1. Create proxy applications
2. Deploy proxies to an API Gateway runtime

* Mule Gateway: CloudHub or a customer-hosted runtime
* Flex Gateway: fast, lightweight, environment-independent

1. Specify throttling, security, and other policies
2. Specify tiers with different types of access
3. Approve, reject, or revoke access to APIs by clients
4. Promote managed APIs between environments Review analytics

* **Restricting access to the API :**

1. Use API Manager to manage access to API proxies

* Define SLA tiers
* Apply runtime policies

1. The API Gateway enforces the policies.
2. API Autodiscovery is a mechanism that enables a deployed Mule application to

* Download policies from API Manager
* Act as its own proxy

1. There are out-of-the box policies for many common use cases like Rate limiting, Spike control, Security, etc..
2. You can define custom policies (using XML and YAML) .
3. You can apply multiple policies and set the order.
4. You can define automated policies to comply with common requirements. It Requires a MuleSoft-hosted control plane.
5. A Service Level Agreement tier defines the number of requests that can be made per time frame to an API, Request approval can be set to automatic (free) or manual (for tiers that cost $).

* **Enforcing access to APIs using SLA Tiers :**

1. To enforce, apply an SLA based rate limiting policy.
2. SLA based policies require all applications that consume the API to

* Register for access to a specific tier, from an API portal in private or public Exchange
* Pass their client credentials in calls made to the API
* **Requesting access to SLA Tier :**

1. If an API has an SLA-based policy, a Request access button appears in API portal.
2. To request access, developer must belong to the Anypoint Platform organization and be logged in.
3. When developers request access, they must

* Register/add an app to their Anypoint Platform account
* Select a tier
* **Approving SLA Tier access request :**

1. For tiers with manual approval, emails are sent to the Organization Administrators when developers request access for applications.
2. Organization Administrators can review the applications in API Manager and approve, delete, or revoke requests.

* **Adding client ID enforcement to API Specification :**

1. You need to add client ID enforcement to the API spec for the

* REST connector that is created for the API to enforce the authentication
* Required headers to automatically show up in the API console so you don't have to manually add them for every call

1. Instructions are in the RAML snippet for a policy in API Manager.

VIEWING DATA IN MULE ANYPOINT STUDIO :

* **Viewing event data at design time :**

1. At design time in Anypoint Studio using DataSense

* In the DataSense Explorer
* In the Transform Message component
* When writing expressions using auto-completion

1. DataSense is the ability to proactively discover metadata from internal and external resources

* Keeps you from having to manually discover info about the data
* Facilitates transformations by providing DataWeave expected input or output
* **DataSense Notification and Indicator :**

1. If DataSense is timing out, a notification will appear
2. A red dot DataSense indicator will also appear in the lower-left corner
3. Clicking on this indicator allows for checking on the DataSense process and restarting it if necessary

* **Viewing event data at run time :**

1. At run time

* In the client when making a request
* For deployed applications, in the log files
* In the Anypoint Studio console by using a Logger
* In Anypoint Studio using the Visual Debugger, most comprehensive way
* **View event data by logging it :**

1. Add a Logger component to a flow and view its output
2. Logged values are displayed

* For an application run from Anypoint Studio with embedded runtime, in the Console view
* For applications deployed to CloudHub or customer-hosted runtimes, in the log files

DEBUGGING APPLICATIONS WITH THE MULE DEBUGGER :

1. Can add breakpoints to processors and step through the application

* Watch event properties and values
* Watch and evaluate DataWeave expressions
* Can be managed from the breakpoints list

1. By default, Debugger listens for incoming TCP connections on localhost port 6666

* Can change this in a project's run configuration

Use the Set Payload transformer to set the payload. Set in the HTTP Listener properties view > Responses. Set in the HTTP Request properties view > General.

DATAWEAVE EXPRESSION LANGUAGE :

1. A Mule-specific expression and transformation language
2. Can be used to access and evaluate the data in the payload, attributes, and variables of a Mule event
3. Accessible and usable from all event processors and global elements
4. Is used to modify the way the processors act upon the event such as routing
5. Case-sensitive
6. Easy to use with auto-complete everywhere

* **Types of DataWeave expressions :**

1. Standalone scripts : Can be generated using the Transform Message graphical editor
2. Inline expressions :

* Are used to dynamically set the value of properties in an event processor or global configuration element
* Are enclosed in #[] and many places in the product offer an expression mode button
* **DataWeave functions :**

1. Functions are packaged in modules Functions in the Core module are imported automatically into DataWeave scripts
2. For functions with 2 parameters, an alternate syntax can be used

Example : #[contains(payload, "max")]

#[payload contains "max"]

* **Creating variables :**

1. Create variables to store metadata for the Mule event
2. Use the Set Variable transformer to create a variable
3. In expressions, reference as vars

Example : #[vars.foo]

FLOWS AND SUBFLOWS :

* **Flows vs subflows :**

1. Flows can have their own error handling strategy, subflows cannot Flows without event sources are sometimes called private flows.
2. Subflows are executed exactly as if the processors were still in the calling flow.

* **Break up flows into separate flows and subflows :**

1. Makes graphical view more intuitive when we don't want long flows that go off screen
2. Makes XML code easier to read
3. Enables code reuse
4. Provides separation between an interface and implementation
5. Makes them easier to test

* **Creating flows and subflows :**

1. Several methods :

* Add a new scope: Flow or Sub Flow
* Drag any event processor to the canvas - creates a flow
* Right-click processor(s) in canvas and select Extract to

1. Use Flow Reference component to pass events to other flows or subflows. Here, Metadata is automatically propagated and can be customized.
2. Variables persist through all flows unless the event crosses a transport boundary.

VM CONNECTOR :

* **Passing events between flows using asynchronous queues :**

1. When using Flow Reference, events are passed synchronously between flows.
2. You may want to pass events asynchronously between flows to

* Achieve higher levels of parallelism in specific stages of processing
* Allow for more-specific tuning of areas within a flow's architecture
* Distribute work across a cluster
* Communicate with another application running in the same Mule domain
* Implement simple queuing that does not justify a full JMS broker
* This can be accomplished using the VM connector
* **Using the VM connector :**

1. Use the connector for intra and inter application communication through asynchronous queues
2. Add the VM module to the project
3. Configure a global element configuration
4. Specify a queue name and type

* Queues can be transient or persistent
* By default, the connector uses in-memory queues
* Transient queues are faster, but are lost in the case of a system crash
* Persistent queues are slower but reliable

1. Use operations to publish and/or consume events

CONFIGURATION FILES :

* **Separating apps into multiple configuration files :**

Just as we separated flows into multiple flows, we also can separate configuration files into multiple configuration files. Monolithic files are difficult to read and maintain. Separating an application into multiple configuration files makes code

1. Easier to read
2. Easier to work with
3. Easier to test
4. More maintainable

* **Encapsulating global elements in a configuration file :**

If you reference global elements in one file that are defined in various, unrelated files, it can be confusing and makes it hard to find them. A good solution is to put most global elements in one configuration file and all the rest of the files reference them. If a global element is specific to and only used in one file, it can make sense to keep it in that file.

* **Creating multiple applications :**

You are also not going to want all your flows in one application/project. Separate functionality into multiple applications to allow managing and monitoring of them as separate entities, use different, incompatible JAR files. Run more than one application at a time in Anypoint Studio by creating a run configuration.

* **Sharing global elements between applications :**
* A domain project can be used to share global configuration elements between applications, which lets you

1. Ensure consistency between applications upon any changes, as the configuration is only set in one place
2. Expose multiple services within the domain on the same port
3. Share the connection to persistent storage
4. Call flows in other applications using the VM connector

* Only available for customer-hosted Mule runtimes, not on CloudHub or RTF
* The general process

1. Create a Mule Domain Project and associate Mule applications with a domain
2. Add global element configurations to the domain project

APPLICATION PROPERTIES :

1. Provide an easier way to manage connector properties, credentials, and other configurations
2. Replace static values
3. Are defined in a configuration file either in a .yaml file or a .properties file
4. Are implemented using property placeholders
5. Can be encrypted
6. Can be overridden by system properties when deploying to different environments

* **Defining application properties :**

1. Create a YAML properties file in the src/main/resources folder config.yaml
2. Define properties in the hierarchical YAML file

* db:

port: "3306"

user: "mule"

1. Create a Configuration properties global element

* **Using application properties :**

1. In global element configurations and event processors

${db.port}

1. In DataWeave expressions

{port: Mule::p('db.port')}

* **Overriding property values in different environments :**

1. Use system properties to override property values when deploying an application to a different environment (like dev, qa, production),
2. Set system properties (JVM parameters) from

* Anypoint Studio in Run > Run Configurations > Arguments
* The command line for a standalone Mule instance

mule -M-Ddb.database=training2 -M-Ddb.password=mule2

ORGANIZING MULE PROJECT FILES :

* **Examining the folder structure for a Mule project :**

1. The names of folders indicate what they should contain
2. src/test folders should contain files only needed at development time

* Like schema and example files for metadata types, sample data for transformations
* They are not included in the application JAR when it is packaged
* **In Mule 4, Mule applications are Maven projects :**

1. Maven is a tool for building and managing any Java-based project that provides

* A standard way to build projects
* A clear definition of what a project consists of
* An easy way to publish project information
* A way to share JARs across several projects

1. Maven manages a project's build, reporting, and documentation from a central piece of information - the project object model (POM)
2. A Maven build produces one or more artifacts, like a compiled JAR. Each artifact has a group ID (usually a reversed domain name, like com.example.foo), an artifact ID (just a name), and a version string.

* **The POM (Project Object Model) :**

Is an XML file that contains info about the project and configuration details used by Maven to build the project including

* Project info like its version, description, developers, and more
* Project dependencies
* The plugins or goals that can be executed

META DATA :

* **Defining metadata :**

It is often beneficial to define metadata for an application

1. For the output structures required for transformations
2. For the output of operations that can connect to data sources of different structures like the HTTP Request connector
3. For the output of connectors that are not DataSense enabled and do not automatically provide metadata about the expected input and output

* **Ways to access the Metadata Editor :**

1. From the Transform Message component
2. From the Metadata tab in the properties view for most event processors
3. From a project's menu in the Package Explorer

The metadata is stored in application-types.xml in src/main/resources.